<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Welcome, researchers!</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Welcome, students!</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>News</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>The Aalto environment</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>Data management</td>
<td>99</td>
</tr>
<tr>
<td>6</td>
<td>Triton</td>
<td>115</td>
</tr>
<tr>
<td>7</td>
<td>Aalto Research Software Engineering</td>
<td>317</td>
</tr>
<tr>
<td>8</td>
<td>Scientific computing</td>
<td>341</td>
</tr>
<tr>
<td>9</td>
<td>Training</td>
<td>373</td>
</tr>
<tr>
<td>10</td>
<td>About</td>
<td>447</td>
</tr>
</tbody>
</table>
This site contains documentation about scientific and data-intensive computing at Aalto and beyond. It is targeted towards Aalto researchers, but has some useful information for everyone. The data management section is useful even to non-computational researchers.

*Aalto Scientific Computing* maintains these pages with the help of the Aalto community. [twitter] We consist of Science-IT (HPC, the Triton cluster), certain department IT's, and other friends. *You can join us.*
Welcome to Aalto, researchers. Aalto has excellent resources for you, but it can be quite hard to know of them all. These pages will provide a good overview of IT services for researchers for you (focused on computation and data-intensive work, including experimental work).

See also:
- These aren’t generic IT instructions - ITS has an introduction for staff somewhere (but apparently not online).
- IT Services for Research is the comprehensive list of researcher-oriented IT services available (compared to this which is a starting tutorial)

1.1 Aalto service units

Understanding all the Aalto services can be quite confusing. Here are some of the key players:

- **Department IT**: Only a few departments (mainly in SCI) have their own IT staff. Others have people such as laboratory managers which may be able to provide some useful advice. Known links: CS, NBE, PHYS, Math.

- **Science-IT**: Overlaps with SCI department IT groups. They run the Triton cluster and support scientific computing. Their services may be used throughout the entire university, but support is organized from the departments which fund them. The core Science-IT departments are CS, NBE, and PHYS. Science-IT runs a weekly *SciComp garage*, where we provide hands on support for anything related to scientific computing. This site [scicomp.aalto.fi](http://scicomp.aalto.fi) is the main home, read more about us on the *about page*.

- **Aalto IT Services (ITS)**: Provides central IT infrastructure. They have a “Services for Research” group, but it is less specialized than Science-IT. ITS may be the first place to contact if not in the School of Science. Their infrastructure is used in all schools including SCI, and the base on which everyone builds. Their instructions are on [aalto.fi](http://aalto.fi), but most importantly the already-mentioned IT Services for Research page. Contact via servicedesk.

- **Aalto Research Services**: Administrative-type support. Provides support for grantwriting, innovation and commercialization, sponsored projects, legal services for research, and research infrastructures. (In 2019 a separate “innovation services” split from the previous “research and innovation services”).

- **CSC** is the Finnish academic computing center (and more). They provide a lot of basic infrastructure you use without knowing it, as well as computing and data services to researchers (all for free). [research.csc.fi](http://research.csc.fi)

Also, currently Aalto has information scattered on websites everywhere:

- [aalto.fi](http://aalto.fi) is the normal homepage, but doesn’t have much practical information for researchers. As of late 2018, information from inside and into is supposed to move here. The new site is well known for being hard to use (sorry, we can’t do anything about that). This site is “not designed to have a logical structure and instead, you are expected to search for information” (actual quote). Some pages get more information if you log in, and there is no indication of which ones. In general, unless you know what you are looking for, don’t expect to find anything here without extensive work.
Aalto scientific computing guide

- **inside.aalto.fi** is the former typical “official” staff documentation area, but now gone since 2019 with all information moved to aalto.fi. It wasn’t in search engines, but once you found a page it wasn’t easy to find other information.

- **into.aalto.fi** is the student official information. It used to contain a lot of duplicate information to inside but was public, so people would end up there.

- **wiki.aalto.fi** is obviously the Aalto wiki space. Anyone can make a space here, and many departments’ internal sites are here. Searching can randomly find useful information. Most sites aren’t publically searchable.

- **scicomp.aalto.fi** is where you are now. Scicomp was started by the Science-IT team from the Triton (HPC cluster) documentation, and scicomp is slowly taking over from our departments’ research IT instructions. Now it is our general guidance to researchers and the best place to find information on research and scientific computing - as opposed to general “staff computing” you find other places.

### 1.2 End-user systems

Aalto provides computers to it’s employees, obviously. Whether it is an Aalto wide managed system or standalone depends on your department policies. If it’s standalone, you are on your own. If managed, login is through your Aalto account. You can get laptop or desktop, and Linux, Mac, or Windows.

Desktops are connected directly to the wired networks and are typically preferred by researchers using serious data or computation. Linux desktops have fast and automatic access to all of the university data storage systems, including Triton and department storage. They also have a wide variety of scientific software already available (and somewhat similar to Triton). We have some limited instructions and pointers to the main instructions for **mac** and **windows** computers.

Managed laptops are usable in and out of the Aalto networks.

On both managed desktops and laptops you can become a “primary user” which allows you to install needed software that is found from the official repositories. Additionally, in some cases, Workstation Administrator (wa.) account can be given which close to normal root/Administrator account with some limitations. The “primary user” is widely accepted and recommended by Aalto ITS to all users while wa. accounts are regulated by the department policies or Aalto ITS.

### 1.3 Computing

Having a valid Aalto account you have two primary options: workstations and Triton. The Aalto workstations have basic scientific software installed. From the workstations, you can use the **HTCondor** distributed computing framework.

Most demanding computing at Aalto is performed on **Triton**, the Aalto high performance computing cluster. It is a fairly standard medium-sized cluster, and it’s main advantage is the close integration into the Aalto environment: it shares Aalto accounts, its data storage (2PB) is also available on workstations, and has local support. If you need dedicated resources, you can purchase them and they can be managed by Science IT team as part of Triton so that you get dedicated resources and can easily scale to the full power of Triton. Triton is part of the Finnish Grid and Cloud Infrastructure. Triton is the largest publically known computing cluster in Finland after the CSC clusters. Triton provides a web-based interface via **JupyterHub**. To get started with Triton, **request access**, check the tutorials sequence, and you’ll learn all you need.

**CSC** (the Finnish IT Center for Science) is a government-owned organization which provides a lot of services, most notably huge HPC clusters, data, and IT infrastructure services to the academic sector. All of their services are free to the academic community (paid directly by the state of Finland). They also coordinate the Finnish Grid and Cloud Infrastructure. They have the largest known clusters in Finland.
1.4 Data

Data management isn’t just storage: if data is just put somewhere, you get a massive mess and data isn’t usable in even 5 years. Funders now require “data management plans”. Thus data management is not just a hot topic, it’s an important one. We have a whole section on data, and also there are higher level guides from Aalto. If you just want to get something done, you should start with our Aalto-specific guideline for Science-IT data storage (used in CS, NBE, PHYS) - if you follow our plan, you will be doing better than most people. If you have specific questions, there is an official service email address you can use (see the Aalto pages), or you can ask the Science-IT team.

Aalto has many data storage options, most free. In general, you should put your data in some centralized location shared with your group: if you keep it only on your own systems, the data dies when you leave. We manage data by projects: a group of people with shared access and a leader. Groups provide flexibility, sharing, and long-term management (so that you don’t lose or forget about data every time someone leaves). You should request as many projects as you need depending on how fine-grained you need access control, and each can have its own members and quota. You can read about the storage locations available and storage service policy.

Triton has 2PB of non-backed up data storage on the high-performance Lustre filesystem. This is used for large active computation purposes. The Triton nodes have an incredible bandwidth to this and it is very fast and parallel. This is mounted by default at Science-IT departments, and can be by default in other departments too.

Aalto provides “work” and “teamwork” centralized filesystems which are large, backed up, snapshotted, shared: everything you may want. Within the Science-IT departments, Science-IT and department IT manages it and provides access. For other schools/departments, both are provided by Aalto ITS but you will have to figure out your school’s policies yourself. It’s possible to hook this storage into whatever else you need over the network. (In general, “work” is organized by the Aalto hierarchy, while “teamwork” is flatter. If you consider yourself mainly Aalto staff who fits in the hierarchy, work is probably better. If you consider yourself a research who collaborates with whoever, teamwork is better.) Teamwork instructions

CSC provides both high-performance Lustre filesystems (like Triton) and archive systems. CSC research portal.

In our data management section, we provide many more links to long-term data repositories, archival, and so on. The fairdata.fi project is state-supported and has a lot more information on data. They also provide some data storage focused on safety and longer-term storage (like IDA), though they are not very used at Aalto because we provide such good services locally.

Aalto provides, with Aalto accounts, Google Drive (unlimited, also Team Drives), Dropbox (unlimited), and Microsoft OneDrive (5TB). Be aware that once you leave Aalto, this data will disappear!

1.5 Software

Triton and Aalto Linux workstations come with a lot of scientific software installed, with in the Lmod system. Triton generally has more. If you need something, it can be worth asking us first to install it for everyone.

If you are the primary user of a workstation, you can install Ubuntu packages yourself (and if you aren’t, you should ask to be marked as primary user). If you use Triton or are in a Science-IT department, it can be worth asking Science-IT about software you need - we are experts in this and working to simplify the mess that scientific software is. Windows workstations can have things automatically installed, check the windows page.

Triton and Aalto workstations have the central software available, currently for laptops you are on your own except for some standard stuff.

On Triton and Linux workstations, type module spider $name to search for available software. We are working to unify the software stack available on Triton and Aalto workstations so that they have all the same stuff.

ITS has a software and licenses (FI) page, and also a full list of licenses (broken link, missing on new page). There is also https://download.aalto.fi/.
CSC also has a lot of software. Some is on CSC computers, some is exported to Triton.

1.6 Starting a project

Each time you start a project, it’s worth putting a few minutes into planning so that you create a good base (and don’t end up with chaos in a few years). We don’t mean some grant, we mean a line of work with a common theme, data, etc.

• Think about how you’ll manage data. It’s always easy to just start working, but it can be worth getting all project members on the same page about where data will be stored and what you want to happen to it in the end. Having a very short thing written will also help a lot to get newcomers started. The “practical DMP” section here can help a lot - try filling out that A4 page to consider the big sections.

• Request a data group (see above) if you don’t already have a shared storage location. This will keep all of your data together, in the same place. As people join, you can easily give them access. When people leave, their work isn’t lost.
  – If you already have a data group that is suitable (similar members), you can use that. But there’s no limit to the number of projects, so think about if it’s better to keep things apart earlier.
  – Mail your department IT support and request a group. Give the info requested at the bottom of data outline page.
  – In the same message, request the different data storage locations, e.g. scratch, project, archive. Quotas can always be increased later.

1.7 Training

Of course you want to get straight to research. However, we come from a wide range of backgrounds and we’ve noticed that missing basic skills (computer as a tool) can be a research bottleneck. We have constructed a multi-level training plan so that you can find the right courses for your needs. These courses are selected by researchers for researchers, so we make sure that everything is relevant to you.

Check our upcoming training page for a list of upcoming courses. If you do anything computational or code-based at all, you should consider the twice-yearly CodeRefinery workshops (announced on our page). If you have a Triton account or do high-performance computing or intensive computing or data-related tasks, you should come to the Summer (3 days) or Winter (1 day) kickstart, which teaches you the basics of Triton and HPC usage (we say it is “required” if you have a Triton account).

1.8 Other notes

Remember to keep the IT Services for Research page close at hand!

Research is usually collaborative, but sometimes you can feel isolated - either because you are lost in a crowd, or far away from your colleagues. Academic courses don’t teach you everything you need to be good at scientific computing - put some effort into working together with, learning from, and teaching your colleagues and you will get much further.

There are some good cheatsheets which our team maintains. They are somewhat specialized, but useful in the right places.

It can be hard to find your way around Aalto, the official campus maps and directions are known for being confusing confusing. Try UsefulAaltoMap instead.
See also:

Primary information is at Aalto’s IT Services for Students page, which focuses on basic services. This focuses on students in computing and data intensive programs.

Welcome to the Aalto! We are glad you are interested in scientific computing and data. scicomp.aalto.fi may be useful to you, but is somewhat targeted to research usage. However, it can still serve as a good introduction to resources for scientific and data-intensive computing at Aalto if you are a student. This page is devoted to resources which are available to students.

If you are involved in a research group or doing research for a professor/group leader, you are a researcher! You should acquaint yourself with all information on this site, starting with Welcome, researchers! and use whatever you need.

General IT instructions can be found at https://www.aalto.fi/en/it-help. There used to be some on into.aalto.fi, but these are gone now. There also used to be a 2-page PDF introduction for students, but it also seems to be gone from online. IT Services for Students is now the best introduction.

2.1 Accounts

In general, your Aalto account is identical to that which researchers have — the only difference is that you don’t have an departmental affiliation.

2.2 Getting help

As a student, the ITS servicedesks are the first place to go for help. The site https://www.aalto.fi/en/it-help is the new central site for IT instructions.

This site, https://scicomp.aalto.fi, is intended for research scientific computing support but has a few page useful to you.
2.3 Computation

As a student, you have access to various light computational resources which are suitable for most courses that need extra power:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paniikki computer lab</td>
<td>Linux workstations, GPUs, software via modules</td>
</tr>
<tr>
<td>Other computer labs</td>
<td>Workstations, different OSs</td>
</tr>
<tr>
<td>Shell servers</td>
<td>Via ssh, software via modules, overcrowded. Brute and Force are for computation, others not.</td>
</tr>
<tr>
<td>JupyterHub</td>
<td>Basic software, in web browser</td>
</tr>
<tr>
<td>Remote desktop</td>
<td>Windows and Linux</td>
</tr>
<tr>
<td>Own computers</td>
<td>Software at <a href="https://download.aalto.fi">https://download.aalto.fi</a></td>
</tr>
</tbody>
</table>

The Jupyter service at https://jupyter.cs.aalto.fi is available to everyone with an Aalto account. It provides at least basic Python and R software; we try to keep it up to date with the things people need most for courses that use programming or data.

The shell servers brute and force are for light computing, and generally for students. You may find them useful, but can often be overloaded. Light computing shell servers. Learn how to launch Jupyter notebook on there.

For GPU computing, the Paniikki Linux computer lab (map) has GPUs in all workstations. Software is available via module spider $name to search and module load $name to load (and the module anaconda has Python, tensorflow, etc.). Read the Paniikki cheatsheet here. The instructions for Aalto workstations sort of apply there as well. The software on these machines is managed by the Aalto-IT team. This is the place if you need to play with GPUs, deep learning, etc, and helps you transition to serious computing on large clusters.

A new (2018) remote desktop service is available at https://vdi.aalto.fi (instructions). This provides Windows and Linux desktops and is designed to replace the need for computer classrooms with special software installed. You can access it via a web browser or the VMware Horizon client. More VDI Windows workstations are also available at http://mfavdi.aalto.fi/.

The use of Triton is for research purposes and students can’t get access unless you are affiliated with a research project or (in very rare cases), a course makes special arrangements.

2.4 Data storage

Aalto home directories have a 100GB quota, and this is suitable for small use. Note that files here are lost once you leave Aalto, so make sure you back up.

The IT Services for Research page contains some other cloud services which may be useful for data storage. Of the cloud services, note that everyone at Aalto can get an unlimited Google Drive account through the Aalto Google Apps service: instructions. Your Aalto Google account will expire once you are no longer affiliated, so your files here will become inaccessible.
2.5 Software

ITS has a software and licenses (FI) page, and also a full list of licenses. There is also [http://download.aalto.fi/](http://download.aalto.fi/). Various scientific software can be found for your own use via the Aalto software portals.

The Lmod (module) system provides more software on brute/force and in Paniikki. For example, to access a bunch of scientific Python software, you can do `module load anaconda`. The researcher-focused instructions are here, but like many things on this site you may have to adapt to the student systems.

Common software:

<table>
<thead>
<tr>
<th>Package</th>
<th>Installation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td><code>module load anaconda</code> on Linux</td>
</tr>
<tr>
<td>Tensorflow etc packages</td>
<td>same as Python, in Paniikki</td>
</tr>
</tbody>
</table>

2.6 Other notes

It can be hard to find your way around Aalto, the official campus maps and directions are known for being confusing. Try [UsefulAaltoMap](http://www.usefulaalto.com) instead.

Do you have suggestions for this page? Please leave an issue on Github (make sure you have a good title that mentions the audience is students, so we can put the information in the right place). Better yet, send a pull request to us yourself.
7-9/06/2021 New Triton user? Join our course on how to use Triton and HPC https://scicomp.aalto.fi/training/scip/summer-kickstart/

Please note that during the summer our daily garage has a slightly different schedule. Furthermore, if you are into research software, join our Research Software Hour Twitch show, usually every Thursday at 21:30 (Helsinki time).

3.1 News

7-9/06/2021 New Triton user? Join our course on how to use Triton and HPC https://scicomp.aalto.fi/training/scip/summer-kickstart/

Please note that during the summer our daily garage has a slightly different schedule. Furthermore, if you are into research software, join our Research Software Hour Twitch show, usually every Thursday at 21:30 (Helsinki time).

3.1.1 News archive

10/05/2021 CodeRefinery online workshop starts today. Tune in for git intro part 1. If you did not register, you can watch via Twitch: https://www.twitch.tv/coderefinery

01/04/2021 April fools’ . . . NOT: no jokes but instead a reminder that we have new courses starting in April “Hands on Data Anonymization” and “Software Design for Scientific Computing”. More info and registration links at https://scicomp.aalto.fi/training/

19/03/2021 Linux Shell Scripting starts next week! There is still time to register at: https://scicomp.aalto.fi/training/scip/shell-scripting/

15/02/2021 We have a new login node and new software versions on Triton for: abinit, anaconda, cuda, julia, and quantum espresso. Read more at our issue tracker. We recommend following the issue tracker for live updates from us and from our users too!

14/01/2021 Save the date: 29 January 2021: Crash course on Data Science workflows at Aalto + Linux terminal basics in preparation for 1-2 February 2021: Triton Winter Kickstart. Registration link can be found within the course pages. Kickstart course is highly recommended to new Triton HPC users.

10/12/2020 We are updating and consolidating our tutorials and guidelines on https://scicomp.aalto.fi website. There might be temporary broken links, please let us know if you spot anything that does not look as it should. Please note that the next Research Software Hour on https://twitch.tv/RSHour will be on Thursday 17/12 at 21:30 Helsinki time. A special episode about Advent of Code 2020.

02/12/2020 This week Research Software Hour on https://twitch.tv/RSHour will happen during the day, straight from the https://nordic-rse.org/ meeting! 13:30 Helsinki time: All you wanted to know about the Rust programming language! Past episodes at Research Software Hour.

19/11/2020 Our course on Matlab Basics finishes today. Videos from the course will be uploaded to the Aalto Scientific Computing YouTube channel. See the course webpage for more info.

10/11/2020 Our course on Matlab Basics starts today. See the course webpage for more info.

29/10/2020 Today at 21:30 Helsinki time, join us for another live episode of Research Software Hour on https://twitch.tv/RSHour Tonight: git-annex to version control your data and HPC cluster etiquette.

26/10/2020 Tomorrow day 4 of our online CodeRefinery workshop. Materials are available here https://coderefinery.github.io/2020-10-20-online and if you did not register, you can watch it live at https://www.twitch.tv/coderefinery.

21/10/2020 Today day 2 of our online CodeRefinery workshop. Materials are available here https://coderefinery.github.io/2020-10-20-online and if you did not register, you can watch it live at https://www.twitch.tv/coderefinery.

20/10/2020 Today day 1 of our online CodeRefinery workshop. Come and learn about version control, jupyter, documentation. Materials are available here https://coderefinery.github.io/2020-10-20-online and if you did not register, you can watch it live at https://www.twitch.tv/coderefinery.

19/10/2020 Today ***"Triton users group meeting"***, come and hear about the future of Triton/ScienceIT/Aalto Scientific Computing, exciting news on new services, new hardware (GPUs!), and anything related to Aalto Scientific Computing.

16/10/2020 Today the fourth an last part of our course on Data analysis workflows with R and Python. You can watch it on CodeRefinery Twitch channel.

14/10/2020 Today our course on Data analysis workflows with R and Python continues. You can watch it on CodeRefinery Twitch channel. Please note that the last part of the course is on Friday 16/10/2020.

13/10/2020 Tomorrow our course on Data analysis workflows with R and Python continues. You can watch it on CodeRefinery Twitch channel.

06/10/2020 Today is Tuesday, however Research Software Hour has now moved from Tuesdays to Thursdays. Tune in on Twitch on Thursday October 15 at 21:30 (Helsinki time) to watch live the next episode.

05/10/2020 Today starts our Data analysis workflows with R and Python. You can watch it on CodeRefinery Twitch channel.

29/09/2020 - Join us tonight (21:30 Helsinki time), for Research Software Hour, a one hour interactive discussion with Radovan Bast and Richard Darst. Tonight how to organise research software projects and other tips to keep track of notes, tools, etc.

28/09/2020 – Friendly reminder that you can still register for our Data analysis workflows with R and Python. Link to registration is here. Also save the date: Mon 19/10/2020 at 14:00 “Triton users group meeting”, come and hear about the future of Triton/ScienceIT/Aalto Scientific Computing, exciting news on new services, new hardware (GPUs!), and anything related to Aalto Scientific Computing. More details coming soon.

25/09/2020 – Friendly reminder that you can still register for our Data analysis workflows with R and Python. Link to registration is here.

24/09/2020 – Join our informal chat about research software on zoom at 10:00: RSE activities in Finland. Today is also the SciComp garage day focused on HPC/Triton issues: daily garage.

23/09/2020 – Last day of our course on “Python for Scientific Computing” covering packaging and binder. It can also be watched live on CodeRefinery Twitch if you did not have time to register.

22/09/2020 – Join us tonight (21:30 Helsinki time), for Research Software Hour, a one hour interactive discussion with Radovan Bast and Richard Darst. Tonight we cover command line arguments and running things in parallel. You can watch RSH past episodes on YouTube to get an idea of the topics covered.
21/09/2020 – This week is the last week of our course on “Python for Scientific Computing” You can re-watch the lessons on CodeRefinery Twitch channel

14/09/2020 – Our course on “Python for Scientific Computing” has started today. It can also be watched live on CodeRefinery Twitch if you did not have time to register.

08/09/2020 – “Research Software Hour” will start on 22/09/2020. RSH is an interactive, streaming web show all about scientific computing and research software. You can watch past episodes at the RSH video archive on youtube.

xx/09/2020 – We started a small News section to keep users up to date and avoid missing important things coming up. Check our trainings coming in October and November. Join our daily garage if you have issues to discuss related to computing or data management.
CHAPTER FOUR

THE AALTO ENVIRONMENT

Aalto provides a wide variety of support for scientific computing. For a summary, see the IT Services for Research page. For information about data storage at Aalto, see the section on data management below.

4.1 Aalto tools

For more services provided at the Aalto level, see the IT Services for Research page.

4.1.1 Aalto account

Extension to Aalto account and email

Aalto account expiration is bound to staff or student status. Account closes one week after the affiliation to Aalto university ends. Expiration is managed completely by Aalto IT Services, and department IT staff is not able to extend Aalto accounts.

If extension to account is needed, this may be achieved with visitor contract. The contract requires host information, so you should contact your supervisor who (if accepting your request) contacts HR with needed details to prepare the official visitor contract.

4.1.2 Aalto Linux

See also:

https://linux.aalto.fi/ provides official information on Aalto Linux for all Aalto. This page is a bit focused on the Science-IT departments, but also useful for everyone.

Aalto Linux is provided to all departments in Aalto. Department IT co-maintains this, and in some departments provides more support (specifically, CS, NBE, PHYS and Math at least). It contains a lot of software and features to support scientific computing and data. Both laptop and desktop setups are available.

This page is mainly about the Linux flavor in CS/PHYS/NBE and partly Math, co-managed by these departments and Science-IT. Most of it is relevant to all Aalto, though.
Basics

- **Aalto home directory.** In the Aalto Ubuntu workstations, your home directory will be your Aalto home directory. That is, the same home directory that you have in Aalto Windows machines and the Aalto Linux machines, including shell servers (kosh, talta, lyta, brute, force).

- **Most installations have Ubuntu 16.04 or 18.04, 20.04 is coming soon.**

- **A pretty good guide is available at https://linux.aalto.fi.**

- **Login is with Aalto credentials.** Anyone can log in to any computer. Since login is tied to your Aalto account, login is tied to your contract status. Please contact HR if you need to access systems after you leave the university or your account stops working due to contract expiration.

- **All systems are effectively identical, except for local Ubuntu packages installed. Thus, switching machines is a low-cost operation.**

- **Systems are centrally managed using puppet. Any sort of configuration group can be set up, for example to apply custom configuration to one group’s computers.**

- **Large scientific computing resources are provided by the Science-IT project. The compute cluster there is named Triton.** Science-IT is a school of science collaboration, and its administrators are embedded in NBE, PHYS, CS IT.

- **Workstations are on a dedicated network VLAN.** The network port must be configured before it can be turned on and you can’t just assume that you can move your computer to anywhere else. You can request other network ports enabled for personal computers, just ask.

- **Installation is fully automated via netboot.** Once configuration is set up, you can reboot and PXE boot to get a fresh install. There is almost no local data (except the filesystem for tmp data on the hard disks which is not used for anything by default, /1/ below), so reinstalling is a low-cost operation. The same should be true for upgrading, once the new OS is ready you reboot and netinstall. Installation takes less than two hours.

- **Default user interface.** The new default user interface for Aalto Linux is Unity. If you want to switch to the previous default interface (Gnome), before logging in please select “Gnome Flashback (Metacity)” by clicking the round ubuntu logo close to the “Login” input field.

- **Personal web pages.** What you put under ~/public_html will be visible at https://users.aalto.fi/~username. See Data storage.

When requesting a new computer:

- Contact your department IT

- Let us know who the primary user will be, so that we can set this properly.

When you are done with a computer:

- Ensure that data is cleaned up. Usually, disks will be wiped, but if this is important then you must explicitly confirm before you leave. There may be data if you use the workstation local disks (not the default). There is also a local cache ($XDG_CACHE_HOME), which stores things such as web browser cache. Unix permissions protect all data, even if the primary user changes, but it is better safe than sorry. Contact IT if you want wipes.
Laptops

- You can get laptops with Linux on it.
- Each user should log in the first time while connected to the Aalto network. This will cache the authentication information, then you can use it wherever you want.
- Home directories can be synced with the Aalto home directories. This is done using unison. TODO: not documented, what about this?
- If you travel, make sure that your primary user is set correctly before you go. The system configuration can’t be updated remotely.
- Otherwise, environment is like the workstations. You don’t have access to the module system, though.
- If the keychain password no longer works: see FAQ at the bottom.

Workstations

Most material on this page defaults to the workstation instructions.

Primary User

The workstations have a concept of the “primary user”. This user can install software from the existing software repositories and ssh remotely to the desktops.

- **Primary users are implemented as a group with name** `$hostname-primaryuser`. You can check primary user of a computer by using `getent group $hostname-primaryuser` or check your primary-userness with `groups`.
- If you have a laptop setup, make sure you have the PrimaryUser set! This can’t be set remotely.
- **Make sure to let us know about primary users when you get a new computer set up or change computers.** You don’t have to, but it makes it convenient for you.
- It is not currently possible to have group-based primary users (a group of users all have primary user capabilities across a whole set of computers, which would be useful in flexible office spaces). TODO: are we working on this? (however, one user can have primary user access across multiple computers, and hosts can have multiple primary users, but this does not scale well)

Data

See the general storage page for the full story (this is mainly oriented towards Linux). All of the common shared directories are available on department Linux by default.

We recommend that most data is stored in shared group directories, to provide access control and sharing. See the Aalto data page.

You can use the program unison or unison-gtk to synchronise files.
Full disk encryption (Laptops)

All new (Ubuntu 16.04 and 18.04) laptops come with full disk encryption by default (instructions). This is a big deal and quite secure, if you use a good password.

When the computer is first turned on, you will be asked for a disk encryption password. Enter something secure and remember it - you have only one chance. Should you want to change this password, take the computer to an Aalto ITS service desk. They can also add more passwords for alternative users for shared computers. Aalto ITS also has a backup master key. (If you have local root access, you can do this with cryptsetup, but if you mess up there’s nothing we can do).

Desktop workstations do not have full disk encryption, because data is not stored directly on them.

Software

Already available

- Python: module load anaconda (or anaconda2 for Python 2) (desktops)
- Matlab: automatically installed on desktops, Ubuntu package on laptops.

Ubuntu packages

If you have PrimaryUser privileges, you can install Ubuntu packages using one of the following commands:

- By going to the Ubuntu Software Center (Applications -> System Tools -> Administration -> Ubuntu Software Centre). Note: some software doesn’t appear here! Use the next option.
- aptdcon --install $ubuntu_package_name (search for stuff using apt search)
- By requesting IT to make a package available across all computers as part of the standard environment. Help us to create a good standard operating environment!

The module system

The command module provides a way to manage various installed versions of software across many computers. This is the way that we install custom software and newer versions of software, if it is not available in Ubuntu. Note that these are shell functions that alter environment variables, so this needs to be repeated in each new shell (or automated in login).

- See the Triton module docs docs for details.
- module load triton-modules will make most Triton software available on Aalto workstations (otherwise, most is hidden).
- module avail to list all available package.
- module spider $name to search for a particular name.
- module load $name to load a module. This adjusts environment variables to bring various directories into PATH, LD_LIBRARY_PATH, etc.
- We will try to keep important modules synced across the workstations and Triton, but let us know.

Useful modules:

- anaconda and anaconda2 will always be kept up to date with the latest Python Anaconda distribution, and we’ll try to keep this in sync across Aalto Linux and Triton.
• **triton-modules**: a metamodule that makes other Triton software available.

**Admin rights**

Most times you don’t need to be an admin on workstations. Our Linux systems are centrally managed with non-standard improvements and features, and 90% of cases can be handled using existing tools:

Do you want to:

• **Install Ubuntu packages**: Use `aptdcon --install $package_name` as primary user.

• **This website tells me to run `sudo apt-get` to install something. Don’t, use the instructions above.**

• **This website gives me some random instructions involving `sudo` to install their program. These are not always a good idea to run, especially since our computers are networked, centrally managed, and these instructions don’t always work. Sometimes, these things can be installed as a normal user with simple modifications. Sometimes their instructions will break our systems. In this case, try to install as normal user and then send a support request first. **If none of these work and you have studied enough to understand the risk, you can ask us. Make sure you give details of what you want to do.**

• **I need to change network or some other settings. Desktops are bound to a certain network and settings can’t be changed, users can’t be managed, etc.**

• **It’s a laptop: then yes, there are slightly more cases you need this, but see above first.**

• **I do low-level driver, network protocol, or related systems development. Then this is a good reason for root, ask us.**

If you do have root and something goes wrong, our help is limited to reinstalling (wiping all data - note that most data is stored on network drives anyway).

If you do need root admin rights, you will have to fill out a form and get a new wa account, then Aalto has to approve. Contact your department IT to get the process started.

**Remote access to your workstation**

If you are primary user, you can ssh to your own workstation from certain Aalto servers, including at least taltta. See the [remote access page](#).

**More powerful computers**

There are different options for powerful computing.

First, we have desktop Linux workstations that are more powerful than normal. If you want one of these, just ask. It includes a medium-power GPU card. You can buy a more powerful workstation if you need, but...

Beyond that, we recommend the use of Triton rather than constructing own servers which will only be used part-time. You can either use Triton as-is for free, or pay for dedicated hardware for your group. Your own hardware as part of Triton means that you can use all Triton and even CSC if you need with little extra work. You could have your own login node, or resources as part of the queues.

Triton is Aalto’s high-performance computing cluster. It is not a part of the department Linux, but is heavily used by researchers. You should see the main documentation at the [Triton user guide](#), but for convenience some is reproduced here:

• Triton is CentOS (compatible with the Finnish Grid and Cloud Infrastructure), while CS workstations are Ubuntu. So, they are not identical environments, but we are trying to minimize the differences.
  
  – Since it is is part of FGCI, it is easy to scale to more power if needed.
We will try to have similar software installed in workstation and Triton module systems.

The paths `/m/`/dept/ are designed to be standard across computers

The `project` and `archive` filesystems are not available on all Triton nodes. This is because they are NFS shares, and if someone starts a massively parallel job accessing data from here, it will kill performance for everyone. Since history shows this will eventually happen, we have not yet mounted them across all nodes.

- These are mounted on the login nodes, certain interactive nodes, and dedicated group nodes.
- TODO: make this actually happen.

Triton was renewed in 2016 and late 2018.

All info in the triton user guide

Common problems

Network shares are not accessible

If network shares do not work, there is usually two things to try:

- Permission denied related problems are usually solved by obtaining new Kerberos ticket with command `kinit`
- If share is not visible when listing directories, try to `cd` to that directory from terminal. Shares are mounted automatically when they are accessed, and might not be visible before you try to change to the directory.

Graphical User Interface on Aalto CS Linux desktop is sluggish, unstable or does not start

- 1. Check your disk quota from terminal with command `quota`. If you are not able to log in to GUI, you can change to text console with CTRL+ALT+F1 key combo and log in from there. GUI login can be found with key combo CTRL+ALT+F7.
- 2. If you are running low on quota (blocks count is close quota), you should clean up some files and then reboot the workstation to try GUI login again.

Enter password to unlock your login keyring

You should change your Aalto password in your main Aalto workstation. If you change the password through e.g. https://password.aalto.fi, then your workstation’s password manager (keyring) does not know the new password and requests you to input the old Aalto password.

If you remember your old password, try this:

1. Start application Passwords and Keys (“seahorse”)
2. Click the “Login” folder under “Passwords” with right mouse button and select “Change password”
3. Type in your old password to the opening dialog
4. Input your current Aalto password to the “new password” dialog
5. Reboot the workstation / laptop

If changing password didn’t help, then try this:
• Then instead of selecting the “change password” from the menu behind right mouse key select “delete” and reboot the workstation. When logging in, the keyring application should use your logging key automatically.

In linux some process is stuck and freezez the whole session

You can kill a certain (own) process via text console.

How do I use eJournals, Netmot and other Aalto library services from home?

There is a weblogin possibility at Aalto Library. After this, all library provided services are available. There are links for journals (nell) and netmot. Or use VPN which should already be configured.

Rsync complains about Quota, even though there is plenty left.

The reason usually is that default rsync -av tries to preserve the group. Thus, there is wrong group in the target. Try using rsync -rlptDxzv --chmod=Dg+s <source> <target>. This will make group setting correct on /scratch/ etc and quota should then be fine.

Quota exceeded or unable to write files to project / work / scratch / archive

Most likely this is due to wrong Linux filesystem permissions. Quota is set per group (e.g. braindata) and by default file go to the default group (domain users). If this happens under some project, scratch etc directory it will complain about “Disk quota exceeded”.

In general this is fixed by admins by setting the directory permissions such that all goes ok automatically. But sometimes this breaks down. Some programs often are responsible for this (rsync, tar for instance).

There are two easy ways to fix this

• In terminal, run the command `find . -type d -exec chmod g+rwxs {} \;` under your project directory. After this all should be working normally again.
• If it’s on scratch or work, see the Triton quotas page
• Contact NBE-IT and we will reset the directory permissions for the given directory

I cannot start Firefox

There are two reasons for this.

1. Your network home disk is full

```bash
# Go to your user dir
cd ~
# Check disk usage
du -sh *
```

The sum should be less than the max quota which is 100GB (as of 2020). If your disk is full then delete something or move it to a local directory, /l/.
2. Something went wrong with your browser profile

If you get an error like “The application did not identify itself”, following might solve the issue.

Open terminal,

```
firefox -P -no-remote
```

This will launch Firefox and ask you to choose a profile. Note that when you delete a profile you delete passwords, bookmarks and etc. So it’s better to create a new profile, migrate bookmarks and delete the old one.

4.1.3 Aalto Mac

This page describes the Aalto centrally-managed Mac computers, where login is via Aalto accounts. If you have a standalone laptop (one which does not use your Aalto account), some of this may be relevant, but for the most part you are on your own and you will access your data and Aalto resources via Remote Access.

More instructions: https://inside.aalto.fi/display/ITServices/Mac

Basics

In the Aalto installations, login is via Aalto account only.

- When you get a computer, ask to be made primary user (this should be default, but it’s always good to confirm). This will allow you to manage the computer and install software.
- The first time you login, you must be on an Aalto network (wired or aalto wifi) so that the laptop can communicate with Aalto servers and get your login information. After this point, you don’t need to be on the Aalto network anymore.
- Login is via your Aalto account. The password stays synced when you connect from an Aalto netowrk.

Full disk encryption

This must be enabled per-user, using FileVault. You should always do this, there is no downside. On Aalto-managed laptops, install “Enable FileVault disk encryption” (it’s a custom Aalto thing that does it for you). To do this manually, “Settings → Privacy → enable File Vault.”

Data

You can mount Aalto filesystems by using SMB. Go to Finder → File or Go (depending on OS) → Connect to Server → enter the smb:// URL from the data storage pages.

You can find more information at For generic ways of accessing, see Remote Access. For Aalto data storage locations see Data storage, and for the big picture of where and how to store data see Data: outline, requesting space, requesting access.

The program AaltoFileSync is pre-installed and can be used to synchronize files. But you basically have to set it up yourself.
Software

.dmg files

If you are the primary user, in the Software Center you can install the program “Get temporary admin rights”. This will allow you to become an administrator for 30 minutes at a time. Then, you can install .dmg files yourself. This is the recommended way of installing .dmg files.

Aalto software

There is an application called “Managed software center” pre-installed (or “Managed software update” in older versions). You can use this to install a wide variety of ready-packaged software. (ITS instructions).

Homebrew

Homebrew is a handy package manager on Macs. On Aalto Macs, you have to install Brew in your home dir. Once you install brew, you can easily install whatever you may need.

First install Xcode through Managed Software Centre (either search Xcode, or navigate through Categories -> Productivity -> Xcode).

```
# Go to wherever you want to have your Brew and run this
mkdir Homebrew && curl -L https://github.com/Homebrew/brew/tarball/master | tar xz -C ~ --strip 1

# This is a MUST!!!
echo "export PATH=\$PATH:${pwd}/Homebrew/bin" >> ~/.zprofile

# Reload the profile
source ~/.zprofile

# Check if brew is correctly installed.
which brew    # /Users/username/Homebrew/bin/brew
```

Older versions of MacOS (pre Mojave) use bash as the default shell, therefore you need to setup the environment differently:

```
echo "export PATH=\$PATH:${pwd}/Homebrew/bin" >> ~/.bash_profile

# Reload the profile
source ~/.bash_profile
```

Admin rights

The “Get temporary admin rights” program described under .dmg file installation above lets you get some admin rights - but not full sudo and all.

You don’t need full admin rights to install brew.

If you need sudo rights, you need a workstation admin (wa) account. Contact your department admin for details.
CS Mac backup service

The CS department provides a full clone-backup service for Aalto-installation mac computers. Aalto-installation means the OS is installed from Aalto repository.

We use Apple Time Machine. Backup is wireless, encrypted, automatic, periodic and can be used even outside the campus using the Aalto VPN. It is “clone” because we can restore your environment in its entirety. You can think of it as a snapshot backup (though it isn’t). We provide twice the space of your SSD; your Mac has 250GB of space, you get 500GB of backup space. If you would like to enroll in the program please pay a visit to our office, T-talo A243.

Encryption

We provide two options for encryption:

1. You set your own encryption key and only you know it. The key is neither recoverable nor resettable. You lose it, you lose your backup.
2. We set it on behalf of you and only we know it.

Restore

With Time Machine you have two options for restore.

1. Partial
   - You can restore file-by-file. Watch the video,
2. Complete restore
   - In case your Mac is broken, you can restore completely on a new Mac. For this, you must visit us.

Trouble-shooting

Can’t find the backup destination

This happens because either 1). you changed your Aalto password or 2). the server is down. Debug in the following manner,

```
# Is the server alive?
ping timemachine.cs.aalto.fi

# If alive, probably it’s your keychain.
# Watch the video below.

# If dead, something’s wrong with the server.
# Please contact CS-IT.
```
Corrupted backup

This is an unfortunate situation with an unknown reason. We take a snapshot of your backup. Please contact CS-IT.

Common problems

Insane CPU rampage by UserEventAgent

It is a mysterious bug which Apple hasn’t solved yet. We can reinstall your system for you.

4.1.4 Aalto Windows

This page describes the Aalto centrally-managed Windows computers, where login is via Aalto accounts. If you have a standalone laptop (login not using Aalto account), some of this may be relevant, but for the most part you will access your data and Aalto resources via Remote Access.

More instructions: https://inside.aalto.fi/display/ITServices/Windows

Basics

In the Aalto installations, login is via Aalto account only.

- You must be on the Aalto network the first time you connect.

Full disk encryption

Aalto Windows laptops come with this by default, tied to your login password. To verify encryption, find “BitLocker” from the start menu and check that it is on.

Note, that on standalone installations, you can do encryption by searching “TrueCrypt” in programs - it is already included.
Data

This section details built-in ways of accessing data storage locations. For generic ways of accessing remotely, see *Remote Access*. For Aalto data storage locations, see *Data storage* and *Data: outline, requesting space, requesting access*.

Your home directory is automatically synced to some degree.

You can store local data at `C:\LocalUserData\User-data\<yourusername>`. Note that this is not backed up or supported. For data you want to exist in a few years, use a network drive. It can be worth making a working copy here, since it can be faster.

Software

Aalto software

There is a [Windows software self-service portal](#) which can be used to install some software automatically.

Installing other software

To install most other software, you need to apply for a workstation admin (wa) account. Contact your department IT to get the process started.

Common problems

4.1.5 Data storage

This page outlines the storage options available for your data. There are many options available, some provided by CSIT, some provided by Aalto IT Services, and some provided by Science IT. For clarity, this page describes them all, so that you can have an easy reference.

When starting a new project, please first consider the big picture of good Research Data Management: See the [general data management pages here](#) and Aalto’s page. On Aalto’s page, there are links to solutions for Opening, Collaborating and Archiving. Our department’s resources are just one part of that.

This page is currently a bit Linux-centric, because Linux is best supported.

Other operating systems: Windows and OSX workstations do not currently have any of these paths mounted. In the future, project and archive may be automatically mounted. You can always remote mount via sshfs or SMB. See the [remote access](#) page for Linux, Mac, and Windows instructions for home, project, and archive. In OSX, there is a shortcut in the launcher for mounting home. In Windows workstations, this is Z drive. On your own computers, you may need to use `AALTO\username` as your username for any of the SMB mounts.

Laptops: Laptops have their own filesystems, including home directories. These are not backed up automatically. Other directories can be mounted as described on the [remote access](#) page.
## Summary table

This table lists all available options in Science-IT departments, including those not managed by departments. In general, **project** is for most research data that requires good backups. For big data, use **scratch**. Request **separate projects** when needed to keep things organized.

<table>
<thead>
<tr>
<th>Filesystem (Linux)</th>
<th>Quota</th>
<th>Backups?</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>home /u/.../$username/unix</td>
<td>100 GiB</td>
<td>yes, $HOME/../.snapshot/</td>
<td>Used for personal and non-research files</td>
</tr>
<tr>
<td>project /m/$dept/project/$project/</td>
<td>per-project, up to 100s of GiB</td>
<td>Yes, hourly/daily/weekly. (.snapshot)</td>
<td></td>
</tr>
<tr>
<td>archive /m/$dept/archive/$project/</td>
<td>per-project, up to 100s of GiB</td>
<td>Yes, hourly/daily/weekly. + off-site tape backups. (.snapshot)</td>
<td></td>
</tr>
<tr>
<td>scratch /m/$dept/scratch/$project/</td>
<td>per-project, 2 PiB available</td>
<td>RAID6, but no backups.</td>
<td>Don’t even think about leaving irreplaceable files here! Need Triton account.</td>
</tr>
<tr>
<td>work (Triton) /m/$dept/work/$username/</td>
<td>200 GB default</td>
<td>RAID6, but no backups.</td>
<td>Same as scratch. Need Triton account.</td>
</tr>
<tr>
<td>local /l/$username/</td>
<td>usually a few 100s GiB available</td>
<td>No, and destroyed if computer reinstalled.</td>
<td>Directory needs to be created and permissions should be made reasonable (quite likely 'chmod 700 /l/$USER', by default has read access for everyone!) Space usage: <code>du -sh /l/</code>. Not shared among computers.</td>
</tr>
<tr>
<td>tmpfs /run/user/$uid/</td>
<td>local memory</td>
<td>No</td>
<td>Not shared.</td>
</tr>
<tr>
<td>custom solutions</td>
<td></td>
<td></td>
<td>Contact us for special needs, like sensitive data, etc.</td>
</tr>
</tbody>
</table>
General notes

• The table below details the types of filesystems available.
• The path /m/$dept/ is designed to be a standard location for mounts. In particular, this is shared with Triton.
• The server magi is magi.cs.aalto.fi and is for the CS department. Home directory is mounted here without kerberos protection but directories under /m/ need active kerberos ticket (that can be acquired with ‘kinit’ command). taltta is taltta.aalto.fi and is for all Aalto staff. Both use normal Aalto credentials.
• Common problem: The Triton scratch/work directories are automounted. If you don’t see it, enter the full name then tab complete and it will appear. It will appear after you try accessing with the full name.
• Common problem: These filesystems are protected with Kerberos, which means that you must be authenticated with Kerberos tickets to access them. This normally happens automatically, but they expire after some time. If you are using systems remotely (the shell servers) or have stuff running in the background, this may become a problem. To solve, run kinit and it will refresh your tickets.

Filesystem list

• home: your home directory
  – Shared with the Aalto environment, for example regular Aalto workstations, Aalto shell servers, etc.
  – Should not be used for research work, personal files only. Files are lost once you leave the university.
    * Instead, use project for research files, so they are accessible to others after you leave.
  – Quota 100 GiB.
  – Backups recoverable by $HOME/../../.snapshot/ (on linux workstations at least).
  – SMB mounting: smb://home.org.aalto.fi/
• project: main place for shared, backed-up project files
  – /m/$dept/project/$project/
  – Research time storage for data that requires backup. Good for e.g. code, articles, other important data. Generally for small amount (10s-100s GiB) of data per project.
  – This is the normal place for day to day working files which need backing up.
  – Multi user, per-group.
  – Quotas: from 10s to 100s of GiB
  – Quotas are not designed to hold extremely large research data (TiBs). Ideal case would be 10s of GiB, and then bulk intermediate files on scratch.
  – Weekly backup to tape (to recover from major failure) + snapshots (recover accidentally deleted files). Snapshots go back:
    * hourly last 26 working hours (8-20)
    * daily last 14 days
    * weekly last 10 weeks
    * Can be recovered using .snapshot/ within project directories
  – Accessible on magi/taltta at the same path.
  – SMB mounting: smb://tw-cs.org.aalto.fi/project/$group/
• archive:
- `/m/$dept/archive/$project/`

- For data that should be kept accessible for 1-5 years after the project has ended. Alternatively a good place to store a copy of a large original data (backup).
- This is practically the same as project, but retains snapshots for longer so that data is ensured to be written to tape backups.
- This is a disk system, so does have reasonable performance. (Actually, same system as project, but separation makes for easier management).
- Quotas: 10s to 1000s of GiB
- Backups: same as project.
- Accessible on magi/talta at the same path.
- SMB mounting: `smb://tw-cs.org.alto.fi/archive/$group/`

• **scratch**: large file storage and work, not backed up (Triton).

- `/m/$dept/scratch/$group/`

- Research time storage for data that does not require backup. Good for temporary files and large data sets where the backup of original copy is somewhere else (e.g. archive).
- This is for massive, high performance file storage. Large reads are extremely fast (1+ GB/s).
- This is a lustre file system as part of triton (which is in Keilaniemi).
- Quotas: 10s to 100s of TiB. The university has 2 PB available total.
- In order to use this, you must have a triton account. If you don’t, you get “input/output error” which is extremely confusing.
- On workstations, this is mounted via NFS (and accessing it transfers data from Keilaniemi on each access), so it is not fast on workstations, just large file storage. For high performance operations, work on triton and use the workstation mount for convenience when visualizing.
- This is RAID6, so is pretty well protected against single disk failures, but not backed up at all. It is possible that all data could be lost. Don’t even think about leaving irreplaceable files here. CSC actually had a problem in 2016 that resulted in data loss. It is extremely rare (decades) thing, but it can happen. (still, it’s better than your laptop or a drive on your desk. Human error is the greatest risk here).
- Accessible on magi/talta at the same path.
- SMB mounting: `smb://data.triton.aalto.fi/scratch/$dept/$dir/`. (Username may need to be AALTO\yourusername.)

• **Triton work**: personal large file storage and work (Triton)

- `/m/$dept/work/$username/`

- This is the equivalent of scratch, but per-person. Data is lost once you leave.
- Accessible on magi/talta at the same path.
- SMB mounting: `smb://data.triton.aalto.fi/work/$username`. (Username may need to be AALTO\yourusername.)
- Deleted six months after your account expires.
- Not to be confused with Aalto work (see below).

• **local**: local disks for high performance
– You can use local disks for day to day work. These are not redundant or backed up at all. Also, if your computer is reinstalled, all data is lost.
– Performance is much higher than any of the other network filesystems, especially for small reads. Scratch+Triton is still faster for large reads.
– If you use this, make sure you set UNIX permissions to restrict the data properly. Ask if you are not sure.
– If you store sensitive data here, you are responsible for physical security of your machine (as in no one taking a hard drive). Unix permissions should protect most other cases.
– When you are done with the computer, you are also responsible for secure management/wiping/cleanup of this data.
– See the note about disk wiping under Aalto Linux (under “when you are done with your computer”). IT should do this, but if it’s important you must mention it, too.

• **tmpfs**: in-memory filesystem
  – This is a filesystem that stores all data in memory. It is extremely high performance, but extremely temporary (lost on each reboot). Also shares RAM with your processes, so don’t use too much and clean up when done.
  – TODO: are these available everywhere?

• **webhome**: web space for users.aalto.fi
  – This is the space for users.aalto.fi space can be accessed from the public_html link in your home directory.
  – This is not a real research filesystem, but convenient to note here.
  – Quota (2020) is 5 GiB. (/m/webhome/webhome/)
  – https://users.aalto.fi/~USER/

• **triton home**: triton’s home directories
  – Not part of departments, but documented here for convenience
  – The home directory on Triton.
  – Backed up daily.
  – Not available on workstations.
  – Quota: 1 GB
  – Deleted six months after your account expires.

• **Aalto work**: Aalto’s general storage space
  – /work/$deptcode on Aalto workstations and servers.
  – Not often used within Science-IT departments: we use project and archive above, which are managed by us and practically equivalent. You could request space from here, but expect less personalized service.
  – Aalto home directories are actually here now.
  – You may request storage space from here, email the Aalto servicedesk and request space on work. The procedures are not very well established.
  – Data is snapshotted and backed up offsite for disaster recovery.
  – Search https://it.aalto.fi for “work.org.aalto.fi” for the latest instructions.
  – SMB mounting via smb://work.org.aalto.fi
• **Aalto teamwork**: Aalto’s general storage space
  
  – Not used directly within Science-IT departments: we have our own direct interfaces to this, and project and archive directories are actually here.
  
  – For information on getting teamwork space (outside of Science-IT departments), contact servicedesk.
  
  – Teamwork is unique in that it is arbitrarily extensible, and you may buy the space from the vendor directly. Thus, you can use external grant money to buy storage space here.
  
  – SMB mounting via smb://teamwork.org.aalto.fi

**Quota errors**

Use the `quot` command to see your quota. If you have scratch or work mounted, the quota command will hang and produce errors. For now, check your scratch/work quotas on Triton.

The scratch and work directories do quotas by unix group, and there is a strange error about quota exceeded that you may get sometimes when the unix group of the file or directory is wrong. See the full information at Quotas and summary below. You may have to fix this on Triton if the things below don’t work.

• Symptoms: “Quota exceeded” when you are trying to make a new file in scratch or work directory.

• Root cause: quotas are by groups, and if a directory is not setgroupid (chmod g+s), then files being created will have a different group (with no quota for that location), thus quota exceeded by default. This often happens when you copy a directory from one place to another, and then later try to make new files in that directory.

• Solution: chmod g+s $directory or find $directory -type d -exec chmod g+s {} \
  (you don’t want to make regular files g+s mode).

### 4.1.6 Data: outline, requesting space, requesting access

**Note:** Need a place to store your data? This is the place to look. First, we expect you to read and understand the top information. Then, see the instructions at bottom.

This page is about how to handle data - not the raw storage part, which you can find at data storage. Aalto has high-level information on research data management, too.

**What is data management?**

Data management is much more than just storage. It concerns everything from data collection, to data rights, to end-of-life (archival, opening, etc). This may seem far-removed from research practicalities, but funding agencies are beginning to require advanced planning. Luckily, there are plenty of resources at Aalto (especially in SCI), and it’s just a matter of connecting the dots.

Oh, and data management is also important because without data management, data becomes disorganized, you lose track, and as people come and go, you lose knowledge of what you have. Don’t let this happen to you or your group!

Another good starting point is the Aalto research data management pages. These pages can also help with preparing a data management plan.

Data management is an important part of modern science! We are here to help. These pages both describe the resources available at Aalto (via Science-IT), and provide pointers to issues that may be relevant to your research.
Data storage at Aalto SCI (principles and policies)

Note: This especially applies to CS, NBE, and PHYS (the core Science-IT departments). The same is true for everyone using Triton storage. These policies are a good idea for everyone at Aalto, and are slowly being developed at the university level.

Most data should be stored in a **group (project) directory**, so that multiple people can access it and there is a plan for after you leave. Ask your supervisor/colleagues what your group’s existing groups are and where the data is stored. **Work data should always be stored in a project directory, not personal home directories.** See below for how to create or join a group. Home directory data can not be accessed by IT staff, according to law and policy - data there dies when you leave.

All data in group directories is considered accessible to all members (see below). All data stored should be Aalto or research related. Should there be questions, ask. Finnish law and Aalto policies must be followed (in that order), including by IT staff. Should there be agreements with third-parties regarding data rights, those will also be followed by IT staff, but these must be planned in advance.

All data must have an owner and lifespan. We work with large amount of data from many different people, and data without clear ownership becomes a problem. (“ownership” refers to decision-making responsibility, not IPR ownership). Also, there must be a clear successor for when people leave or become unavailable. By default, this is supervisor.

Personal workstations are considered stateless and, unless there is special agreement, could be reinstalled at any time and are not backed up. This should not concern day to day operations, since by default all data is stored on network filesystems.

We will, in principle, make space for whatever data is needed. However, it is required that it be managed well. If you can answer what the data contains, why it’s stored, and how the space is used, and why it’s needed, it’s probably managed well for these purposes.

**Read the full Science-IT data management policy here.**

Information on all physical locations how to use them is on the storage page.

**Groups**

Everywhere on this page, “group” refers to a certain file access group groups (such as a unix group), not an organizational (research) group. They will often be the same, but there can be many more access groups made for more fine-grained data access.

Data is stored in group directories. A group may represent a real research group, a specific project, or specific access-controlled data. These are easy to make, and they should be extensively used to keep data organized. If you need either finer-grained or more wide data access, request that more groups are made.

Please note, that by design all project data is accessible to every member in the group. This means that, when needed, IT can fix all permissions so that all group members can read all data. For limiting the access more fine-grained than these project groups, please have a separate group created. Data in a group is considered “owned and managed” by the group owner on file. The owner may grant access to others and change permissions as needed. Unless otherwise agreed, any group member may also request permissions to be corrected so that everyone in the group has access.

- **Access control is provided by unix groups** (managed in the Aalto active directory). There can be one group per group leader, project, or data that needs isolation. You should use many groups, they make overall management easier. A group can be a sub-group of another.

- **Each group can get its own quota** and filesystem directories (project, archive, scratch, etc). Quota is per-filesystem. Tell us requested quota when you set up a project.

32 Chapter 4. The Aalto environment
– A typical setup would be: one unix group for a research group, with more groups for specific project when that is helpful. If there are fixed multi-year projects, they can also get a group.

- Groups are managed by IT staff. To request a group, mail us with the necessary information (see bottom of page).
- Each group has an owner, quota on filesystems, and some other metadata (see below).
- Group membership is per-account, not tied to employment contracts or HR group membership. If you want someone to lose access to a group you manage, they have to be explicitly removed by the same method they were added (asking someone or self-service, see bottom of page).

- **To have a group created and storage space allocated**, see below.
- **To get added to a group**, see instructions below.
- To see your groups: use the `groups` command or `groups $username`
- To see all members of a group: `getent group $groupname`

**Common data management considerations**

**Organizing data**

This may seem kind of obvious, but you want to keep data organized. Data is always growing in volume and variety, so if you don’t organize it as it is being made, you have no chance of doing it later. Organize by:

- Project
- To be backed up vs can be recreated
- Original vs processed.
- Confidential or not confidential
- To be archived long-term vs to be deleted

Of course, make different directories to sort things. But also the group system described above is one of the pillars of good data organization: sort things by group and storage location based on how it needs to be handled.

**Backups**

Backups are extremely important, not just for hardware failure, but consider user error (delete the wrong file), device lost or stolen, etc. Not all locations are backed up. It is your responsibility to make sure that data gets stored in a place with sufficient backups. Note that personal workstations and mobile devices (laptops) are not backed up.

**Confidential or sensitive data**

**Note:** The following description is written for the CS department, but applies almost equally to NBE and PHYS. This is being expanded and generalized to other department as well. Regardless of your department, these are good steps to follow for any confidential data at Aalto.

**Note:** This meets the requirements for “Confidential” data, which covers most use cases. If you have extreme requirements, you will need something more (but be careful about making custom solutions).
Aalto has some guidelines for classification of confidential information, but they tend to deal with documents as opposed to practical guidelines for research data. If you have data which needs special attention, you should put it in a separate group and tell us when creating the group.

The following paragraph is a “summary for proposals”, which can be used when the CS data security needs to be documented. This is for the CS department, but similar thing can be created for other departments. A longer description is also available.

Aalto CS provides secure data storage for confidential data. This data is stored centrally in protected datacenters and is managed by dedicated staff. All access is through individual Aalto accounts, and all data is stored in group-specific directories with per-person access control. Access rights via groups is managed by IT, but data access is only provided upon request of the data owner. All data is made available only through secure, encrypted, and password-protected systems: it is impossible for any person to get data access without a currently active user account, password, and group access rights. Backups are made and also kept confidential. All data is securely deleted at the end of life. CS-IT provides training and consulting for confidential data management.

If you have confidential data at CS, follow these steps. CS-IT takes responsibility that data managed this way is secure, and it is your responsibility to follow CS-IT’s rules. Otherwise you are on your own:

• Request a new data folder in the project from CS-IT. Notify them that it will hold confidential data and any special considerations or requirements. Consider how fine-grained you would like the group: you can use an existing group, but consider how many people will have access.

• Store data only in this directory on the network drive. It can be accessed from CS computers, see data storage.

• To access data from laptops (Aalto or your own), use network drive mounting, not copying. Also consider if temporary files: don’t store intermediate work or let your programs save temporary files to your own computer.

• Don’t transfer the data to external media (USB drives, external hard drives, etc) or your own laptops or computers. Access over the network.

• All data access should go through Aalto accounts. Don’t send data to others and or create other access methods. Aalto accounts provide central auditing and access control.

• Realize that you are responsible for the day to day management of data and using best practices. You are also responsible for ensuring that people who have access to the data follow this policy.

• In principle, one can store data on laptops or external devices with full disk encryption. However, in this case we does not take responsibility unless you ask us first.you must ask us about this. In general it’s best to try to adapt to the network drive workflow. (Laptop full disk encryption is a good idea anyway).

We can assist in creating more secure data systems, as can Aalto IT security. It’s probably more efficient to contact us first.

Personal data (research data about others, not about you)

“Personal data” is any data concerning an identifiable person. Personal data is very highly regulated (mainly by the Personal Data Act, soon by the General Data Protection Regulation). Aalto has a document that describes what is needed to process personal data for research, which is basically a research-oriented summary of the Personal Data Act. Depending on the type of project, approval from the Research Ethics Committee may be needed (either for publication, or for human interaction. The second one would not usually cover pure data analysis of existing data). Personal data handling procedures are currently not very well defined at Aalto, so you will need to use your judgment.

However, most research does not need data to be personally identifiable, and thus research is made much simpler. Thus, you want to try to always make sure that data is not identifiable, even to yourself using any technique (anonymization). The legal requirement is “reasonable likelihood of identification”, which can include technical and confidentiality measures, but in the end is still rather subjective. Always anonymize before data arrives at Aalto, if possible. Let us know when you have personal data, so we can make a note of it in the data project.
However, should you need to use personal data, the process is not excessively involved beyond what you might expect (informed consent, ethics, but then a notification of personal data file). Contact us for initial help in navigating the issues and RIS for full advice.

**Openness**

Aalto strongly encourages to share the data openly or under controlled access with a goal of 50% data shared by 2020 (see The Aalto RDM pages). In short, Aalto says that you “must” make strategic decisions about openness for the best benefits (which practically probably means you can do what you would like). Regardless, being open is usually a good idea when you can: it builds impact for your work and benefits society more.

Zenodo (https://zenodo.org/) is an excellent platform for sharing data, getting your data cited (it provides a DOI), and control what you share with different policies (https://about.zenodo.org/policies/). For larger data, there are other resources, such as IDA/AVAA provided by CSC (see below).

There are lists of data repositories: r3data, and Nature Scientific Data’s list.

Datasets can and should also be listed on ACRIS, just like papers - this allows you to get credit for them in the university’s academic reporting.

**Data management plans**

Many funders now require data management plans when submitting grants. (Aside from this, it’s useful to do a practical consideration of how you’ll deal with data)

Please see:

- *The DMP section on this site*
- The Aalto data management plan page

**Long-term archival**

Long-term archival is important to make sure that you have ability to access your group’s own data in the long term. Aalto resources are not currently intended for long-term archival. There are other resources available for this, such as

- the EU-funded Zenodo for open published data (embargoed data and closed data is also somewhat supported).
- Finland’s IDA (for large data, closed or open). There are *Aalto-specific instructions for IDA here*.
- There is supposed to be an alternate Finnish digital preservation service coming in 2017, and it’s unclear what the intention of IDA is in light of that.

**Archival when you leave**

Unfortunately, everyone leaves Aalto sometime. Have you considered what will happen to your data? Do you want to be remembered? This section currently is written from the perspective of a researcher, not a professor-level staff member, but if you are a group leader you need to make sure your data will stay available! Science-IT (and most of these resources) are focused on research needs, not archiving a person’s personal research data (if we archive it for a person who has left, it’s not accessible anyway! Our philosophy is that it should be part of a group as described above.). In general, we can archive data as part of a professor’s group data (managed in the group directories the normal ways), but not for individuals.

- Remember that your home directories get removed when your account expires (we think in only two weeks!).

---

**4.1. Aalto tools** 35
• Data in the group directories it won’t be automatically deleted. But you should clean up all your junk and leave only what is needed for future people. Remember, if you don’t take care of it, it becomes extremely hard for anyone else to. The owner of the group (professor) will be responsible for deciding what to do with the data, so make sure to discuss with them and easy for them to do the right thing!

• Make sure that the data is documented well. If it’s undocmented, then it’s unusable anyway.

• Can your data be released openly? If you can release something as open data on a reputable archive site like Zenodo, you can ensure that you will always have access to it. (The best way to back up is to let the whole internet do it for you.)

• For lightweight archival (~5 years past last use, not too big), the archive filesystem is suitable. The data must be in a group directory (probably your professor’s). Make sure that you discuss the plans with them, since they will have to manage it.

• IDA (see above) could be used for archival of any data, but you will have to maintain a CSC account (TODO: can this work, and how?). Also, these projects have to be owned by a senior-level staff person, so you have to transfer it to a group anyway.

• Finland aims to have a long-term archival service by 2017 (PAS), but this is probably not intended for own data, only well-curated data. Anyway, if you need something that long and it isn’t confidential, consider opening it.

Summary of data locations

Below is a summary of the core Science-IT data storage locations.
### Solution | Purpose | Where available? | Backup? | Group management?
---|---|---|---|---
Project | Research time storage for data that requires backup. Good for e.g. code, articles, other important data. Generally for a small amount of data per project. | Workstations, triton login node | Weekly backup to tape (to recover from major failure) + snapshots (recover accidentally deleted files). Snapshots go back • hourly last 26 working hours (8-20) • daily last 14 days - weekly last 10 weeks | yes

Archive | Data which a longer life that project. Practically the same, but better to sort things out early. Also longer snapshot and guaranteed to get backed up to tape. | Workstations, Triton login node. /m/$dept/project/$group. | Same as above | yes

Scratch (group based)/work (per-user) | Large research data that doesn’t need backup. Temporary working storage. Very fast access on Triton. | /m/$dept/$scratch/$groupname, /m/$dept/work/$username. | scratch: yes, work: no

See *data storage* for full info.

### Instructions for storage and access

**Note:** This applies to the Science-IT departments. If you want to apply for storage space from Aalto-IT, you can use these instructions as a model, but their processes are not yet fully developed.

You and users must accept the *data policy* (summary above).

Existing data groups and responsible contacts:

- **CS:** Existing groups and CS-IT (guru) email here
- **NBE:** Existing groups and NBE IT (it-nbe) email here
- **PHYS:**
- **Aalto:** Aalto IT servicedesk

---

4.1. Aalto tools
Aalto scientific computing guide

Requesting to be added to a group

Note: CS department: New! Group owners/managers can add members to their groups self-service. Go to https://domesti.cs.aalto.fi from Aalto networks, over VPN, or remote desktop at https://vdi.aalto.fi, and it should be obvious.

Send an email to the responsible contact (see above) and CC the group owner or responsible person, and include this information:

- Group name that you request to join
- copy and paste this statement, or something similar: “I am aware that all data stored here is managed by the group’s owner and have read the data management policies.”
- Ask the group owner to reply with confirmation.
- Do you need access to scratch or work? If so, you need a Triton account and you can request it now. If you don’t, you’ll get “input/output error” and be very confused.
- Example:
  
  Hi, I (account=omes1) would like to join the group myprof. I am aware that all data stored here is managed by the group’s owner and have read the data management policies. $professor_name, please reply confirming my addition.

Requesting a new group

Send an email to the responsible contact (see above) with the following information. Group owners should be long-term (e.g. professor level) staff.

- Requested group name (you can check the name from the lists below)
- Owner of data (prof or long-term staff member)
- Other responsible people who can authorized adding new members to the group. (they can reply and say “yes” when someone asks to join the group.)
- Who is responsible for data should you become unavailable (default: supervisor who is probably head of department).
- Initial members
- Expiration time (default=max 2 years, extendable. max 5 years archive). We will ping you for management/renewal then.
- Which filesystems and what quota. (project, archive, scratch). See the the storage page.
- Basic description of purpose of group.
- Is there any confidential or personal data (see above for disclaimer).
- Any other notes that CS-IT should enforce, for example check NDA before giving access.
- Example:
  
  I would like to request a new group coolproject. I am the owner, but my postdoc Tiina Tekkari can also approve adding members. (Should I become unavailable, my colleague Anna Algorithmi (also a professor here) can provide advice on what to do with the data)
  
  We would like 20GB on the project filesystem.
  
  This is for our day to day work in algorithms development, we don’t expect anything too confidential.
4.1.7 Science-IT data policy

Note: This was originally developed at CS, but applies to all departments managed by the Science-IT team.

In Aalto, large amounts of data with variety of requirements are being processed daily. This describes the responsibilities of IT support and users with respect to data management.

Everyone should know the summary items below. The full policy is for reference in case of doubts (items in **bold** are things which are not completely obvious).

This policy is designed to avoid the most common problems by advance planning for the majority case. Science-IT is eager to provide a higher level of service for those who need it, but users must discuss with staff. This policy is jointly implemented by department IT and Science-IT.

**Summary for users**

- Do not store research data in home directories, this is not accessible should something happen to you or when you leave. They will be automatically deleted.

- Project directories are accessible to ALL members, files not intended for access by ALL members should be stored in a separate project.

- Workstations and mobile devices are NOT backed up. Directories with backups are noted. It is your responsible to make sure that you store in backed up places. Don’t consider only disk failure, but also user error, loss of device, etc.

- Data stored in project directories is managed by the (professor, supervisor) who owns the directory, and they can make decisions regarding access now and in the future. Any special considerations should be discussed with them.

- Data is not archived or saved for individual users. Data which must be saved should be in a shared project directory with an owner who is still at Aalto. Triton’s individual users data is permanently deleted after 6 months from the expiration date of the user account (Aalto home directories may be deleted even sooner).

- There is no default named security level - of course we keep all data secure, but should you be dealing with legally confidential files, you must ask us.

**Summary for data directory owners (professors or long-term staff)**

- Data in the shared directories controlled by you and you make decisions on it.

- All data within a project is accessible by all members of that project. Make more projects if more granularity is needed.

- Data must have an expiration time, and this is extended as needed. Improperly managed data is not stored indefinitely. If data is long-term archived, it must still have an administrative owner at Aalto who can make decisions about it.

- There must be a succession plan for the data, should the data owner leave or become unavailable to answer questions. By default this is the supervisor or department head. They will make decisions about access, management, and end-of-life.

- We will try to handle whatever data you may need us to. The only prerequisite is that it is managed well. We can’t really define “managed well”, but at least it means you know what it contains and where the space is going.
Detailed policy

This is the detailed policy. The important summary for users and owners is above, but the full details are written below for avoidance of doubts.

Scope

1. This policy concerns all data stored in the main provided locations or managed by Science-IT staff (including its core departments).

Responsibilities

1. In data processing and rules we follow Finnish legislation and Aalto university policies in this order.
2. If there are agreements with a third party organization for data access those rules are honored next. Regarding this type of data we must be consulted first prior to the storing the data.
3. Users are expected to follow all Aalto and CS policies, as well as good security practices.
4. IT is expected to provided a good service, data security, and instruction on best practices.

Storage

1. **All data must have owner and given lifespan. Data cannot be stored indefinitely**, but of course lifespan is routinely extended when needed. There are other long-term archival services.
2. **Work related data should always be stored outside users HOME directory. HOME is meant only for private and non-work related files.** (IT staff is not allowed to retrieve lost research files from a user’s home directory)
3. Other centrally available folders (i.e. Project, Archive, Scratch) than HOME are meant for work related information only.
4. Desktop computers are considered as stateless. They can be re-installed at any point by IT if necessary. Data stored on local workstations is always considered as temporary data and is not backed up. IT support will still try to inform users of changes.
5. Backed-up data locations are listed. **It is the user’s responsibility to ensure that data is stored in backed-up locations as needed. Mobile devices (laptops) and personal workstations are not backed up.**

Ownership, and access rights, and end-of-life

1. Access rights in this policy refer only to file system rights. Other rights (e.g. IPR) to the stored information are not part of this policy.
2. There must be a clear owner and chain of responsibility (successor) for all data (who owns it and can make decisions and who to ask when they leave or become unavailable).
3. For group directories (e.g. project, archive, scratch), file system permissions (possibility to read, write, copy, modify and delete) of these files belongs to group. There is not more granular access, for example single files with more restrictive permissions. Permissions will be fixed by IT on request from group members.
4. The group owner-on-file can make any decisions related to data access, management, or end-of-life.
5. Should a data owner of a group directory become unavailable or unable to answer questions about access, management, or end-of-life, the successor they named may make decisions regarding the data access, including end-of-life. This defaults to their supervisor (e.g. head of department), but should be discussed on data opening.

6. Triton data stored on folders that are not group directories (e.g. the content of /scratch/work/USERNAME or /home/USERNAME) will be permanently deleted after 6 months from the user’s account expiration. Please remember to back up your data if you know that your account is expiring soon. (Note that Aalto home directory data may be removed even earlier)

7. Should researchers need a more complex access scheme, this must be discussed with IT support.

Security/Confidentiality

1. Unless there is a notification, there is no particular guaranteed service level regarding confidential data. However, all services are expected to be as secure as possible and are designed to support confidential data.
2. Should a specific security level be needed, that must be agreed separately.
3. Data stored to the provided storage location is not encrypted at rest.
4. Confidentiality is enforced by file system permissions will be set and access changes will be always confirmed from data owner.
5. All storage medium (hard drives, etc), should be securely wiped to the extend technically feasible at end of life. This is handled by IT, but if it is required it must be handled by the end users.
6. All remote data access should use strong encryption.
7. Users must notify IT support or their supervisor about any security issues or misuse of data.
8. Security of laptops, mobile devices and personal devices is not currently guaranteed by IT support. Confidential data should use centralized IT-provided services only.
9. Users and data owners must take primary responsibility for data security, since technical security is only one part of the process.

Communication

1. Details about centrally provided folders and best practices are available in online documentation.
2. Changes to policy will be coordinated by department management. All changes will at least be announced to data owners, but individual approvals are not needed unless a service level drops.

4.1.8 Remote Access

This page describes remote access solutions. Most of them are provided by Aalto, but there are also instruction for accessing your workstations here. See Aalto Inside for more details.
Linux shell servers

- Department servers have project, archive, scratch, etc mounted, so are good to use for research purposes.
  - CS: magi.cs.aalto.fi: Department staff server (no heavy computing, access to workstations and has file systems mounted, use the kinit command first if project directories are not accessible)
  - NBE: amor.org.aalto.fi, same as above.
  - Math: elliptic.aalto.fi, illposed.aalto.fi, same as above (but no project, archive and scratch directories)

- Aalto servers
  - kosh.aalto.fi, lyta.aalto.fi: Aalto, for general login use (no heavy compting)
  - brute.aalto.fi, force.aalto.fi: Aalto, for “light computing” (expect them to be overloaded and not that useful). If you are trying to use these for research, you really want to be using Triton instead.
  - taltta.aalto.fi: Staff server (access to workstations and has filesystems mounted, but you need to kinit to access them.) that is kind of outdated and different.

- Your home directory is shared on all Aalto shell servers, and that means .ssh/authorized_keys as well.
- You can use any of these to mount things remotely via sshfs. This is easy on Linux, harder but possible on other OSs. You are on your own here. You still need kinit at the same time.
  - The CS filesystems project and archive and Triton filesystems scratch and work are mounted on magi (and taltta.aalto.fi) (see storage).

For any of these, if you can’t access something, run kinit!

VPN / web proxy

To access certain things, you need to be able to connect to the Aalto networks via VPN. This is easy and automatically set up on Aalto computers.

Main Aalto instructions. This section has some quick reference info.

- Generic: OpenConnect/Cisco AnyConnect protocols. vpn.aalto.fi, vpn1.aalto.fi or vpn2.aalto.fi
- Aalto Linux: Status bar → Network → VPN Connections → Aalto TLS VPN.
- Aalto mac: Dock → Launchpad → Cisco AnyConnect Secure Mobility Client
- Aalto windows: Start → Search → AnyConnect
- Personal mac: use Cisco AnyConnect VPN Client
- personal windows: use Cisco AnyConnect VPN Client

For more lightweight things (though not actually easier!), you can use ssh proxy. You are on your own here. ssh -D 8080 $username@kosh.aalto.fi. Configure your web browser or other applications to use a SOCKS5 proxy on localhost:8080 for connections. Remember to revert when done or else you can’t connect to anything (“proxy refusing connections”). The extension FoxyProxy Standard may be useful here, because you can direct only the domains you want through the proxy.
Remote mounting of network filesystems

From Aalto networks (or VPN), you can mount many of the filesystems via SMB. To use this well, you want to get the VPN set up first like mentioned above. (You can also access these filesystems via ssh through the shell servers):

- In all cases, username=aalto username, domain=AALTO, password=Aalto password.
- For NBE/PHYS, replace tw-cs with tw-nbe or tw-phys.
- **Home directories**: smb://home.org.aalto.fi/
- **Project directories**: smb://tw-cs.org.aalto.fi/project/\$name/ (\$name=project name)
- **Archive directories**: smb://tw-cs.org.aalto.fi/archive/\$name/ (\$name=project name)
- **Scratch directories**, see Triton storage.
  - smb://work.org.aalto.fi for Aalto work directories (different than Triton work).

Depending on your OS, you may need to use either your username directly or AALTO\username

On Ubuntu: Files → Left sidebar → Connect to server → use the URLs above. For other Linuxes, you can probably figure it out. (It varies depending on operating system, look around in the finder)

On Mac laptops: Finder → Go menu item → Connect to server → use the URLs above.

On Windows laptops: To do the mounting, Windows Explorer → Computer → Map network drive → select a free letter. smb:// becomes \ (without the smb:), and / becomes \. For example, a full URL could be \tw-cs.org.aalto.fi\project\mygroup. You can also just enter it into the file manager bar.

**Warning:** Must use VPN or Aalto network.

Remember that you must connect to the Aalto VPN first, unless you are on an Aalto laptop on the aalto network.

Accessing you Linux workstation / Triton remotely

- Remote access to desktop workstations is available via the university staff shell servers taltta.aalto.fi or department-specific servers magi.cs.aalto.fi (CS), amor.org.aalto.fi (NBE), elliptic.aalto.fi/illposed.aalto.fi (Math).
- You need to be the PrimaryUser of the desktop in order to ssh to it.
- Remote access to Triton is available from any Aalto shell server: taltta,kosh.aalto.fi, etc.
- SSHing directly to computers using openssh ProxyCommand:
  - Put this in your .ssh/config file under the proper Host line: ProxyCommand ssh taltta.aalto.fi -W \%h:\%p
  - For this to be most useful, you probably want to set up ssh keys, otherwise you will have to enter your password twice.
  - This starts getting beyond the basic level of ssh use, so you may want to read up on ssh keys, Proxy-Command, ControlMaster. It can make your experience much better.
**Remote desktop**

Aalto has remote desktops available at [https://vdi.aalto.fi](https://vdi.aalto.fi) and [http://mfavdi.aalto.fi](http://mfavdi.aalto.fi). This works from any network. There are both Windows and Linux desktops available. They are arranged as virtual machines with the normal desktop installations, so have access to all the important filesystems and all `/m/{dept}/...`

**4.1.9 JupyterHub**

*Note:* This page is about the JupyterHub for light use and teaching, [https://jupyter.cs.aalto.fi](https://jupyter.cs.aalto.fi). The **Triton JupyterHub** for research is documented at [Jupyter](https://jupyter.cs.aalto.fi).

[https://jupyter.cs.aalto.fi](https://jupyter.cs.aalto.fi) is a JupyterHub installation for teaching and light usage. Anyone at Aalto may use this for generic light computing needs, teachers may create courses with assignments using nbgrader. Jupyter has a rich ecosystem of tools for modern computing.

**Basic usage**

Log in with any valid Aalto account. Our environment may be used for light computing and programming by anyone. Your persistent storage has a quota of 1GB. Your data belongs to you, may be accessed from outside, and currently is planned to last no more than one year from last login. You are limited to several CPUs and 1GB memory.

Your notebook server is stopped after 60 minutes of idle time, or 8 hours max time. Please close the Jupyter tab if you are not using it, or else it may still appear as active.

There are some general use computing environments. You will began with Jupyter in the `/notebooks` directory, which is your persistent storage. Your server is completely re-created each time it restarts. Everything in your home directory is re-created, only `/notebooks` is preserved. (Certain files like `.gitconfig` are preserved by linking into `/notebooks/.home/...`)

You begin with a computing server with the usual scipy stack installed, plus a lot of other software used in courses here.

You may access your data as a network drive by SMB mounting it on your own computer - see **Accessing JupyterHub data**. This allows you total control over your data.

JupyterHub has no GPUs, but you can check out the instructions for using the Paniikki GPUs with the JupyterHub data. These instructions are still under development.

Each notebook server is basically a Linux container primarily running a Jupyter notebook server. You may create Jupyter notebooks to interact with code in notebooks. To access a Linux bash shell, create a new terminal - this is a great place to learn something new.

**Accessing JupyterHub data**

Unlike many JupyterHub deployments, your data is *yours* and have many different ways to access it. Thus, we don’t just have jupyter.cs, but a whole constellation of ways to access and do *your work*, depending on what suits you best for each part.

Your data (and as an instructor, your course’s data) can be accessed many ways:

- On jupyter.cs.
- Via network drive *on your own computer as local files.*
On Aalto shell servers (such as kosh.aalto.fi).
- On other department/university workstations.

On Paniikki and Aalto computers

On Paniikki, and the Aalto servers kosh.aalto.fi, lyta.aalto.fi, brute.aalto.fi, and force.aalto.fi (and possibly more), the JupyterHub is available automatically. You can, for example, use the Paniikki GPUs.

Data is available within the paths /m/jhnas/jupyter. The path on Linux servers is also available on the hub, if you want to write portable files.

<table>
<thead>
<tr>
<th>Name</th>
<th>Path on hub</th>
<th>Path on Linux servers</th>
</tr>
</thead>
<tbody>
<tr>
<td>personal notebooks</td>
<td>/notebooks</td>
<td>/m/jhnas/jupyter/u/$nn/$username/</td>
</tr>
<tr>
<td>course data</td>
<td>/coursedata</td>
<td>/m/jhnas/jupyter/course/$course_slug/data/</td>
</tr>
<tr>
<td>course instructor files</td>
<td>/course</td>
<td>/m/jhnas/jupyter/course/$course_slug/files/</td>
</tr>
<tr>
<td>shared data</td>
<td>/m/jhnas/jupyter/shareddata/</td>
<td>/m/jhnas/jupyter/shareddata/</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable seen above</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$username</td>
<td>Your Aalto username</td>
</tr>
<tr>
<td>$nn</td>
<td>The two numbers you see in echo $HOME (the last two digits of your Aalto uid, id)</td>
</tr>
<tr>
<td>$course_slug</td>
<td>The short name of your course.</td>
</tr>
</tbody>
</table>

You can change directly to your notebook directory by using cd /m/jhnas/jupyter/${HOME%/unix}.

You can link it to your home directory so that it’s easily available. In a terminal, run /m/jhnas/u/makedir.sh and you will automatically get a link from ~/jupyter in your home directory to your user data.

Permission denied? Run kinit in the shell - this authenticates yourself to the Aalto server and is required for secure access. If you log in with ssh keys, you may need to do this.

Remote access via network drive

Basic info

<table>
<thead>
<tr>
<th>Name</th>
<th>Network drive path</th>
</tr>
</thead>
<tbody>
<tr>
<td>personal notebooks</td>
<td>smb://jhnas.org.aalto.fi/$username/</td>
</tr>
<tr>
<td>course data</td>
<td>smb://jhnas.org.aalto.fi/course/$course_slug/data/</td>
</tr>
<tr>
<td>course instructor files</td>
<td>smb://jhnas.org.aalto.fi/course/$course_slug/files/</td>
</tr>
<tr>
<td>shared data</td>
<td>smb://jhnas.org.aalto.fi/shareddata/</td>
</tr>
</tbody>
</table>

You can do a SMB mount, which makes the data available as a network drive. You will have the same copy of the data as on the hub - actually, same data, so edits immediately take effect on both places, just like your home directory. You must be on an Aalto network, which for students practically means you must be connected to the Aalto VPN (see vpn instructions) or use an Aalto computer. The “aalto” wifi network does not work unless you have an Aalto computer.

4.1. Aalto tools
· Linux: use “Connect to Server” from the file browser. The path is smb://jhnas.org.aalto.fi/
$\text{username}$. You may need to use AALTO\$\text{username} as your username. If there is separate “domain”
option, use AALTO for domain and just your username for the username.

· Mac: same path as Linux above, “Connect to Server”. Use AALTO\$your\_username as the username.

· Windows: \\jhnas.org.aalto.fi\$\text{username}, and use username AALTO\$your\_username. Win-
dows sometimes caches the username/password for a long time, so if it does not work try rebooting.

You can also access course data and shared data by using jhnas.org.aalto.fi/course/ or jhnas.org.
aalto.fi/shareddata/.

See also:
Mounting network drives in Windows is the same instructions, but for Aalto home directories. Anything there should
apply here, too.

Using GPUs

One problem with our JupyterHub so far is that we don’t have GPUs available. But, because our data is available to
other computers, you can use the Paniikki: Computer Lab For Students GPUs (quite good ones) to get all the power
you need. To do this, you just need to access the Jupyter data on these classroom computers.

Terminal: First, start a terminal. You can navigate to your data following the instructions above: cd /m/jhnas/
jupyter/${HOME%/unix}. From there, navigate to the right directories and do what is needed.

File browser: Navigate to the path /m/jhnas/jupyter/u/$nn/$\text{username}, where $nn$ is the two numbers
you see when you do echo $HOME in a terminal. To open a terminal from a location, right click and select “Open in
Terminal”.

Now that you have the terminal and the data, you can do whatever you want with it. Presumably, you will start Jupyter
here - but first you want to make the right software available. If you course tells you how to do that using an Anaconda
environment, go ahead and do it. (Please don’t go installing large amounts of software like anaconda in the Jupyter
data directories - they are for notebooks and small-medium data.)

Using the built-in anaconda, you can load the Python modules with module load anaconda and start Jupyter
with jupyter notebook:

![Image of terminal with Anaconda modules loaded]

Fig. 1: Note that now, you need to module load anaconda, not anaconda3 like the image shows.

Terms of use

This service must be used according to the general IT usage policy of Aalto university (including no unlawful pur-
poses). It should only be used for academic purposes (but note that self-exploration and programming for own interests
is considered an academic purpose, though commercial purposes is not allowed). For more information, see the Aalto
policies. Heavy non-interactive computational use is not allowed (basically, don’t script stuff to run in the background
when you are not around. If you are using this service is person, it is OK). For research computing, see Triton cluster.
Courses and assignments

Some courses may use the nbgrader system to give and grade assignments. These courses have special entries in the list. If you are a student in such a course, you will have a special environment for that course. Your instructor may customize the environment, or it may one of our generic environments.

If your course is using nbgrader, there are some built-in features for dealing with assignments. Under the Assignment list tab, you can see the assignments for your course (only the course you selected when starting your notebook server). You can fetch assignments to work on them - they are then copied to your personal /notebooks directory. You can edit the assignments there - fill out the solutions and validate them. Once you are done, you can submit them from the same assignment list.

A course may give you access to a /coursedata folder with any course-specific data.

By default, everyone may access every course’s environment and fetch their assignments. We don’t stop you from submitting assignments to courses you are not enrolled in - but please don’t submit assignments unless you are registered, because the instructors must then deal with it. Some courses may restrict who can launch their notebook servers: if you can not see or launch the notebook server for a course you are registered for, please contact your instructor in this case.

Note that the /notebooks folder is shared across all of your courses/servers, but the assignment list is specific to the course you have started for your current session. Thus, you should pay attention to what you launch. Remember to clean up your data sometimes.

Instructors

JupyterHub for instructors

See also:

Main article with general usage instructions: Jupyterhub for Teaching. For research purposes, see Triton JupyterHub.

nbgrader documentation is at https://nbgrader.readthedocs.io/, and is necessary reading to understand how to use it. It is not duplicated here.

jupyter.cs news

Summer/Autumn 2020

• You can now make a direct link that will spawn a notebook server, for example for a course with a slug of testcourse: 'https://jupyter.cs.aalto.fi/hub/spawn?profile=testcourse If the user is already running a server, it will not switch to the new course. Expect some subtle confusion with this. Full info in FAQ and hints.

Basics

The JupyterHub installation provides a way to provide a notebook-based computational environment to students. It is best to not think of this service as a way to do assignments, but as a general light computing environment that is designed to be easy enough to be used for courses. Thus, students should feel empowered to do their own computing and this should feel like a stepping stone to using their own systems set up for scientific computing. Students’ own data is persistent as they go through courses, and need to learn how to manage it themselves. Jupyter works best for project/report type workflows, not lesson/exercise workflows but of course it can do that too. In particular, there is no real possibility for real-time grading and so on.
Optionally, you may use nbgrader (notebook grader) to make assignments, submit them to students, collect them, autograde them, manually grade, and then export a csv/database of grades. From that point, it is up to you to manage everything. There is currently no integration with any other system, except that Aalto accounts are used to login.

What does this mean? Jupyter is not a learning management system (even when coupled with nbgrader), it’s “a way to make computational narratives”. This means that this is not a point and click solution to running courses, but a base to build computations on. In order to build a course, you need to be prepared to do your own scripting and connections using the terminal.

You may find the book Teaching and Learning with Jupyter helpful.

Currently we support Python the most, but there are other language kernels available for Jupyter. For research purposes, see the Triton Jupyter page.

**Limits**

- This is not a captive environment: students may always trivially remove their files and data, and may share notebooks across different courses. See above for the link to isolate-environment with instructions for fixing this.

- We don’t have unlimited computational resources, but in practice we have quite a lot. Try to avoid all students doing all the work right before a deadline and you should be fine, even with hundreds of students.

- There is no integration to any other learning management systems, such as the CS department A+ (yet). The only unique identifier of students is the Aalto username. nbgrader can get you a csv file with these usernames, what happens after that point is up to you.

- There is currently no plagiarism detection support. You will have to handle this yourself somehow so far.

**System environment**

The following is the environment in each Jupyter notebook server exists. This is a normal Linux environment, and you are encouraged to use the shell console to interact with it. In fact, you will need to use the console to do various things, and you will probably need to do some scripting.

Why is everything not a push-button solution? Everyone has such unique needs, and we need to solve all of them. We can only accomplish our goals if people are able to - and do - do their own scripting.

**Linux container**

Each time you launch your server, you get a personal Linux container. Everything (except the data) gets reset each time it stops. From the user perspective, it looks like a normal Linux container. Unlike some setups, we allow students to acknowledge and browse the whole Linux system. (other systems try to hide it, but in reality they can’t stop students from accessing it).
Data

- `/notebooks/` is your per-user area. It’s what you see by default, and is shared among all your courses.
- `/course/` is the course directory (a nbgrader concept). It is available only to instructors. You need to read the nbgrader instructions to understand how this works.
- `/coursedata/` is an optional shared course data directory. Instructors can put files here so that students can access them without having to copy data over and over. Instructors can write here, students can only read. Remember to make it readable to all students: `chmod -R a+rX /coursedata`.
- `/srv/nbgrader/exchange` is the exchange directory, a nbgrader concept but you generally don’t have to worry about it yourself.

Data is available from outside JupyterHub: it is hosted on an Aalto-wide server provided by Aalto. Thus, you can access it on your laptops, on Aalto public shell servers, and more. A fast summary is below, but see Accessing JupyterHub data for the main info.

- From your own laptop: The SMB server `jhnas.org.aalto.fi` path `/vol/jupyter/{course, $username}`.
  - Linux: “Connect to server” from the file browser, URL `smb://jhnas.org.aalto.fi/vol/jupyter`
  - Mac: same as Linux
  - Windows: `\\jhnas.org.aalto.fi\vol\jupyter`.
- Data is available on public Aalto shell servers such as kosh and lyta, at `/m/jhnas/jupyter/`.

Software

For Python, software is distributed through `conda`. You can install your own packages using `pip` or `conda`, but everything is reset when you restart the server. This is sort of by design: a person can’t permanently break their own environment (restarting gets you to a good state), but you have your own flexibility.

You should ask us to install common software which you are your students need, instead of installing it yourself each time. But you should feel free to install it yourself to get your work done until you do that.

Jupyter

Both Jupyter Lab and classic notebooks are installed, along with a lot of extensions. If you need more extensions, let us know. All courses use only the classic notebook interface by default, because the nbgrader web extensions do not work from Lab.

Requesting a course

To get started with a course, please read the below list and describe your needs from the relevant items, and contact guru@cs.aalto.fi. Don’t worry too much about understanding or answering everything perfectly, just let us know what you want to accomplish and we will guide you to what you need.
Course or not?

If all you need is a Python environment to do assignments and projects, you don’t need to request anything special - students can just use the generic servers for their independent computational needs. Students can upload and download any files they need. You could add data to the “shareddata” location, which is available to any user.

You would want a course environment if you want to (distribute assignments to students via the interface) and/or (collect assignments via the interface).

Request template

To make things faster and more complete, copy and paste the below in your email to us (guru@cs.aalto.fi), and edit all of fields (and if anything unclear, don’t worry: send it and a human will figure it out). The format is YAML, by the way (but we can handle the syntax details).

```
name: CS-E0000 Course Name (Year)
uid: (leave blank, we fill in)
gid: (leave blank, we fill in)

# supervisor = faculty in charge of course
# contacts = primary TAs which should also get emails from us.
# manager = (optional) has rights to add other TAs via
domesti.cs.aalto.fi (supervisor is always a manager)
supervisor: teacher.in.charge@aalto.fi
contact: [teacher.in.charge@aalto.fi, head.ta@aalto.fi]
#manager: [can_add.tas@aalto.fi]

datadir: false

# Important dates. But not too important, we can always adjust later.
public_date: 2020-09-08 # becomes visible to students before course
private_date: 2021-01-31 # hidden from students after course
archive_date: 2021-09-01 # becomes hidden from instructors
delete_date: 2021-09-01 # after this, we ask if it can be deleted

# For the course dates itself (just for our reference, not too important)
start_date: 2020-10-01
end_date: 2020-12-15
course_times: Exercise sessions Tuesday afternoons, Deadlines Fridays at 18

# The dates above actually aren't used. These control visibility:
private: false
archive: false

# Internal use, ignore this. The date is the version of software
# you get (your course won't get surprise updates to software after
# that date).
image: [standard, 2020-01-05]
```
Course environment options

When requesting a course, please read the following and tell us your requirements in the course request email, guru@cs.aalto.fi (using the template above). If you are using the hub without a specific course item in the selection list, please let us know at least 3a, 6, 7, and 8 below. You don’t need to duplicate stuff in the YAML above.

Required metadata is:
A course environment consists of (comment on any specifics here):

1. A course directory `/course`, available only to instructors. This comes by default, with a quota of a few gigabytes (combined with coursedata). Note: instructors should manage assignments and so on using git or some other version control system, because the course directory lasts only one year, and is renewed for the next
2. **Software** (optional, recommended to use the default and add what you need) A list of required software, or a docker container containing the Jupyter stack and additional software. By default, we have an image based on the scipy stack and all the latest software that anyone else has requested, as long as it is mutually compatible. You can request additional software, and this is shared among all courses. If you need something special, you may be asked to take our image and extend it yourself. Large version updates to the image are done twice a year during holidays.
   a. (optional) A sample python file or notebook to test that the environment works for your course (which will be made public and open source). We also use use automated testing on our software images, so that we can be sure that our server images still work when they are updated. If you send us a file, either .py or .ipynb, we will add this to our automatic tests. The minimum amount is something like import of the packages you need, a more advanced thing would test the libraries a little bit - do a minimal, quick calculation.

3. **Computational resources** (optional, not recommended) A list of computational resources per image. Default is currently 2GB and 4 processors (oversubscribed). Note that because this is a container, only the memory of the actual Python processes are needed, not the rest of the OS, and memory tends to be quite small.

4. **Shared data directories.** If you have nontrivial data which needs distributing, consider one of these shared directories which saves it from being copied over and over. The notebook directory itself can only support files of up to 2MB to prevent possible problems. If number of students times amount of data is more than a few hundred MB, strongly consider one of the data directories. Read more about this below.
   a. You can use the “shareddata” directory /mnt/jupyter/shareddata. shareddata is available in all notebooks on jupyter.cs.aalto.fi (even outside of your course) and also (eventually) other Aalto servers. This data should be considered public (and have a valid license), even though for now it’s only accessible to Aalto accounts.
   b. /coursedata is only available within your course’s environment (as chosen from the list). coursedata is also assumed to be public to everyone at Aalto, though you have more control over it.
   c. If you use either of these, you can embed the paths directly in your notebooks. This is easy for hub use, but makes it harder to copy the notebooks out of the hub to use on your own computers. This is something we are working on.

Also tell us if you want to join the jupyterhub-courses group to share knowledge about making notebooks for teaching.

**Course data**

See also:

One of the best features of jupyter.cs is powerful data access. See Accessing JupyterHub data

If your course uses data, request a coursedata or shareddata directory as mentioned above. You need to add the data there yourself, either through the Jupyter interface or SMB mounting of data.

If you use coursedata, just start the course environment and instructors should have permissions to put files in there. Please try to keep things organized!

If you use shareddata, ask for permission to put data there - we need to make the directory for you. When asking, tell us the (computer readable short)name of the dataset. In the shareddata directory, you find a README file with some more instructions. All datasets should have a minimum README (copy the template) which makes it minimally usable for others.

In both cases, you need to chmod -R a+rX the data directory so that the data becomes readable to students.

---

4.1. Aalto tools 53
Note: after you are added to relevant group to access the data, it make take up to 12 hours for your account information to be updated so that it can be accessed via remote mounting.

Don’t include large amount of data in the assignment directories - there will be at least four, if not more, copies of data made for every student.

**Data from other courses**

Sometimes, when you are in course A’s environment, you want to access the data from course B. For example, A is the next year’s edition of the course B, and it could be useful to check the old files.

You can access the files for every course which you are an instructor of at the path `/m/jhnas/jupyter/course/`. The *files/* sub-directory is the entire course directory for that course, the same as */course/* in each course image. You can also access the course data directory at */data/* there.

All old courses (for which you are listed as an instructor) are available, but if the course is in the “achived” state, you can’t modify the files.

**Nbgrader basics**

“nbgrader is a tool that facilitates creating and grading assignments in the Jupyter notebook. It allows instructors to easily create notebook-based assignments that include both coding exercises and written free-responses. nbgrader then also provides a streamlined interface for quickly grading completed assignments.” - *nbgrader upstream documentation*

Currently you should read the upstream nbgrader documentation, which we don’t repeat. We have some custom Aalto modifications (also submitted upstream) which are:

**How to use nbgrader**

Read the nbgrader docs! We can’t explain everything again here.

The course directory is */course/*. Within this are *source/*, *release/*, *submitted/*, *autograded/*, and *feedback/*.

**Things which don’t (necessarily) work in nbgrader**

- **Autograde**: if you click the thing, it will work, but is the same as running all your students code *on your own computer* with no security whatsoever. A slightly clever student is able to see other students work (a privacy breach), alter their grades.

- **Feedback**: While it appears to work, it is designed to operate by hashing the contents of the notebook. Thus, if you have to edit the notebook to make it execute, the hash will be different and the built-in feedback distribution will not work.

  Furthermore, don’t expect hidden tests to stay hidden, grading to happen actually automatically, things to be fully automatic, and so on. Do expect a computing environment optimized for learning.

These are just intrinsic to how nbgrader works. We’d hope to fix these sometime, but it will require a more coordinated development effort.
Aalto scientific computing guide

Aalto specifics

- Instructors can share responsibilities, multiple instructors can use the exchange to release/collect files, autograde, etc. Note that with this power comes responsibility - try hard to keep things organized.
- We can have the assignments in /notebooks while providing whole-filesystem access (so that students can also access /coursedata).
- We’ve added some extra security and sharing measures (most of these are contributed straight to nbgrader).
- Join the shared course repository to share knowledge with others

To use nbgrader:

- Request a course as above.
- Read the nbgrader user instructions.
- You can use the Formgrader tab at the top to manage the whole nbgrader process (this automatically appears for instructors). This is the easiest way, because it will automatically set up the course directory, create assignment directories, etc. But, you can use the nbgrader command line, too. It is especially useful for autograding.
- It’s good to know how we arrange the course directory anyway, especially if you want to manage things yourself without Formgrader. The “course directory” (nbgrader term) is /course. The original assignments go in /course/source. The other directories are /course/{nbgrader_step} and, for the most part, are automatically managed.
- New assignments should be in /course/source. Also don’t use + in the assignment filename (nbgrader #928).
- Manage your assignments with git. See below for some hints about how to do this.
- If you ever get permission denied errors, let us know. nbgrader does not support multiple instructors editing the same files that well, but we have tried to patch it in order to do this. We may still have missed some things here.

Version control of course assignments

See also:

Shared jupyterhub-courses version.aalto.fi Gitlab organization to share notebooks and knowledge about running JupyterHub courses.

git is a version control system which lets you track file versions, examine history, and share. We assume you have basic knowledge of git, and here we will give practical tips to use git to manage a course’s files. Our vision is that you should use nbgrader to manage the normal course files, not the students submissions. Thus, to set up the next year’s course, you just clone the existing git repository to the new /course directory. You backup the entire old course directory to maintain the old students work. Of course, there are other options, too.

Create a new git repository in your /course/ directory and do some basic setup:

```
cd /course/
git init
git config core.sharedRepository group
```

You should make a .gitignore file excluding some common things (TODO: maybe more is needed):

```
gradebook.db
release/
submitted/
```

(continues on next page)
The git repository is in /course, but the main subdirectory of interest is the source/ directory, which has the original files, along with whatever other course notes/management files you may have which are under /course. Everything else is auto-generated.

**Autograding**

**Warning:** nbgrader autograde is not secure, because arbitrary student code is run with instructor permissions. Read more from the instructor page.

**nbgrader hints**

These are practical hints on using nbgrader for grades and assignments. You should also see the separate autograding hints page if you use that.

**General**

To export grades, nbgrader export is your central point. It will generate a CSV file (using a custom MyCourses exporter), which you can download, check, and upload to MyCourses.

**If students submit assignments/you use autograding**

See also:

**Autograding**

- In each notebook (or at least the assignment zero), in the top, have a `STUDENT_NUMBER = xxx` which they have to fill in. Asking each student to include the student number in a notebook ensures that you can later write a script to capture it.

**Testing releasing assignments, without students seeing**

Sometimes instructors want to release and collect assignments as a test, while the course is running. To understand how the solution is simpler than “make a new course”, we need to understand what “release” and “collect” do: they just move files around. So, you can just move them to a different place (called the exchange) instead of the one that all students see. Nbgrader docs sure doesn’t do a good job of explaining it, but behind the scenes it’s quite simple, and that simplicity means it’s easy to control if you know what you are up to...

You can equally move your test files around to a test, instructor-only exchange for your own testing. (Actually, this isn’t even needed, you can just copy them directly, test, and put back in the submitted/ directory. But some people want more. So, from the jupyter terminal, we have made these extra aliases:
# Release to test exchange (as instructor):
nbgrader-instructor-exchange release_assignment $assignment_id
# Fetch from test exchange (as instructor, pretending to be a student):
nbgrader-instructor-exchange fetch_assignment $assignment_id
# Submit to test exchange (as instructor, pretending to be a student):
nbgrader-instructor-exchange submit $assignment_id
# Collect to test exchange (as instructor):
nbgrader-instructor-exchange collect $assignment_id

This copies files to and from /course/test-instructor-exchange/, which you can examine and fully control. If you are doing this, you probably need that control anyway. These terms match the normal nbgrader terminology.

There’s no easy way to make a switch between “live exchange” and “instructor exchange” in the web interface, but because of the power of the command line, we can easily do it anyway.

(use type -a nbgrader-instructor-exchange to see just what it does.)

**Known problems**

- The built-in feedback functionality doesn’t work if you modify the submitted notebooks (for example, to make them run). nbgrader upstream limitation.

**Course data**

If you use the /coursedata directory and want the notebook to be usable outside of JupyterHub too, try this pattern:

```python
import os
if 'AALTO_JUPYTERHUB' in os.environ:
    DATA = '/coursedata'
else:
    DATA = 'put_path_here'

# when loading data, always os.path.join(DATA, 'the_file.py')
```

This way, the file can be easily modified to load data from somewhere else. Of course, many variations are possible.

**Converting usernames to emails**

JupyterHub has no access to emails or student numbers. If you do need to link to email addresses, you can do the following.

- ssh to kosh.aalto.fi
- cd to wherever you have exported a csv file with your grades (for example your course directory, cd /m/jhnas/jupyter/course/$course_slug/files/).
- Run /m/jhnas/jupyter/software/bin/username-to-email.py exported_grades.csv - this will add an email column right after the username column. If the username column is not the zeroth (counting from zero), use the $c \$N option to tell it that the usernames are in the $n$th column (zero indexed).
- Save the output somewhere, for example you could redirect it using > to a new filename. A full example:

  ```bash
  /m/jhnas/jupyter/software/bin/username-to-email.py mycourses_export.csv > mycourses_usernames.csv
  ```

4.1. Aalto tools 57
This script is also available on github.

Our scripts and resources

We are soon going to revise all of our instructor info which can be useful to you later.

Autograding

Autograding is sometimes seen as the “holy grail” of using Jupyter for teaching. But you need an appreciation of the level of the task at hand and how to do it.

Autograding

Warning: Running \texttt{nbgrader autograde} is not secure, because arbitrary student code is run with instructor permissions, including access to all instructor files and all other student data. We have designed our own system to make it secure, but we must run it for you. Contact us to use it. If you autograde yourself, you are making a choice to risk privacy of all students (probably violating Finnish law) and the integrity of your grades. This is a long-standing design flaw of nbgrader which we have fixed as best we can.

The secure autograder has to be run manually, by us. Fetch your assignments and contact us in good time.

How deep do you go?

1. Normal Jupyter notebooks, no automation. You might use our JupyterHub to distribute assignments and as a way for students to avoid running their own software, but that’s all.
2. Use nbgrader facilities to generate a student version of assignments, but handle grading yourself ("manually using nbgrader" or via some other system).
3. Full autograding.

You may think “autograding will save me effort”. It \textit{may}, but it will make a whole lot of effort in another way: making your assignment robust to autograding. As someone once said: plan for one day to write an assignment, one week to make it autogradeable, then weeks to make it robust. It doesn’t help that most reference material you can find is about basic programming, not about advanced data science projects.

\textbf{If you use autograding, you have to test your notebooks with many students of different levels. Plan on weeks for this.}

What is autograding?

\texttt{nbgrader} is \textit{not} a fancy thing - it just copies files around. Autograding is \textit{only} running the whole notebook from top to bottom and looking for errors. If there are errors, subtract points. There is not some major platform running in the background that does things \textit{actually} automatically. This is also the primary benefit: a simple system allows your notebooks to be more portable and reusable, and match more closely to real work.
Autograding at Aalto

1. Design your notebook well
2. Collect your notebooks using the nbgrader interface. Don’t click any “autograde” buttons (unless you check the notebook yourself first).
3. Send an email to guru asking specifying your course and assignment and ask for autograding. We will run actually secure autograding on our server soon, and send you a report on what worked or didn’t. Everything gets automatically updated in your environment.
4. Proceed as normal, for example…:
5. If autograding didn’t work for some people, you can check them, modify if needed, and re-run the autograding yourself (since you just checked it).

Designing notebooks for autograding

(please contribute or comment on these ideas)

Check out the upstream autograding hints, which include: hints on writing good test cases, checking if a certain function has been used, checking how certain functions were called, grading plots, and more. But when reading this, not how these examples are simple code - your cases will probably be more complex.

Understand the whole loop of transferring files from you, to student versions, to students, and back. Understand what the loop is not as well. Understand that there isn’t actual automatic autograding.

Have an assignment zero with no content and worth zero (or one) points, which students have to submit just to show they know how the system works (for example, they don’t forget to push “submit”). Maybe it just has some trivial math or programming exercises. This reduces the cognitive load when doing the real assignments.

Design your notebook with a mindset of unit testing. Note that this isn’t the way that notebooks are usually used, though. Functions and testable functions are good. But note that if you put everything in functions, you lose some of the main benefits of notebooks (interactivity made possible by having things in the top-level scope)! Such is life.

Have sufficient tests that are visible to the students, so that they can tell if their answers are reasonable. For example, student-visible tests might check for the shape of arrays, hidden tests check for the actual values. This also ensures that they are approaching it the way you expect.

Similarly, some instructors have found that you must have plenty of structure so that students only have to fill in well-defined chunks, with instructor code before and after. This ensures that students do “the right thing”, but also means that students lose the experience of the “big picture”: loading, preprocessing, and finalization - important skills for the future. Instead, they learn to fill in blanks and no more, no less. So, in this way autograding is a trade-off: more grade able, less realistic.

Within your tests, use variable names that won’t have a conflict (for example, a random suffix like testval_randomstring36456165 instead of testval). This reduces the chance of one of your tests conflicting/overwriting something that the students have added.

Expect students to do everything wrong, and fail in weird ways. Your tests need to be robust.

Consider if your assignment is more open-ended, or there is one specific way to solve it. If it’s more open-ended, consider if you think you’ll be able to make it autogradeable.

nbgrader relies on metadata in order to do the autograding. In order for this to work, the cell metadata needs to be intact. Normally, you can’t even see it for a cell, but it can be affected if: a) cells are copied and pasted to another notebook file (metadata lost, autograding fails), or b) cells are split (metadata duplicated, nbgrader halts then). You should ask students to copy the whole notebook file around when needed.
Public copy of assignments

One disadvantage of a powerful system is that we have to limit access to authorized users. But you shouldn’t let this limit access to your course: there is nothing special about our system, and if you allow others to see your assignments, they can run them themselves. For example, the service https://mybinder.org allows anyone to run arbitrary notebooks from git repositories.

This is also important because your course environment will go away after a few months - do you want students to be able to refer to it later? If so, do the below.

• change to the release/ directory and `git init`. Create a new repo here.

• Manually `git add` the necessary assignment files after they are generated from the source directory. Why do we need a new repo? Because you can’t have the instructor solutions/answers made public.

• Update files (`git commit -a` or some such) occasionally when new versions come out.

• Add a requirements.txt file listing the different packages you need installed for a student to use the notebooks. See the MyBinder instructions for different ways to do this, but a normal Python requirements.txt file is easiest for most cases. On each line, put in a name of a package from the Python Package Index. There are other formats for R, conda, etc, see the page.

• Then, push this release/ repo to a public repository (check mybinder for supported locations). Make sure you don’t ever accidentally push the course repository!

• Then, go to https://mybinder.org/ and use the UI to create a URL for the resources. You can paste this button into your course materials, so that it’s a one-click process to run your assignments.

• Note that mybinder has a limit of 100 simultaneous users for a repository, to prevent too much use for single organization’s projects. This shouldn’t be the first place you direct students for day-to-day work.

• If you have a /coursedata directory, you will have to provide these files some other way. You could put them in the assignment directory and the release/ git repository, but then you’ll need to have notebooks able to load them from two places: /coursedata or .. I’d recommend do this: import os, if os.path.exists('/coursedata'): DATADIR='/coursedata', else: DATADIR='.', ' and then access all data files by os.path.join('DATADIR', 'filename.dat'). This has the added advantage that it’s easy to swap out DATADIR later, too.

FAQ and hints

Shared course repository

There’s a lot to figure out and everyone has to learn by doing. Why not learn from each other? We have a shared jupyterhub-courses repository on version.aalto.fi with a repository for each course. You can browse and learn from how other courses make notebooks, thus saving you time. It also makes it easier for us to help you.

• Decide who are the people to be added to the jupyterhub-courses Gitlab organization (usually those who have long term contracts with Aalto). You can add whoever you want to the your own courses’s repository itself, but organization side should be kept in smaller group so that other TAs won’t get access to courses which they might participate in.

• Setup git for your course. This is something that you might have already done, but here are some general tips for nbgrader specifically.

• After you have gotten an access to the organization, you can create a course in version.aalto and then setup it as a new origin for your git repository: `git remote add new_remote_name {address}` *(Github help)*

• Now you can use to push to this new remote! For example, if your new origin were “gitlab” then `git push gitlab master` would push into version.aalto. Now you should be ready to go!
**Instructions/hints**

- Request a course when you are sure you will use it. You can use the general use containers for writing notebooks before that point.
- Don’t forget about the **flexible ways of accessing** your **course data**.
- The course directory is stored according to the **Science-IT data policy**. In short, all data is stored in group directories (for these purposes, the course is a group). The instructor in change is the owner of the group: this does not mean they own all files, but are responsible for granting access and answering questions about what to do with the data in the long term. There can be a deputy who can also grant access.
- To add more instructors/TAs, go to domesti.cs.aalto.fi and you can do it yourself. You must be connected to an Aalto network. See the Aalto VPN guide for help with connecting to an Aalto network from outside.
- Store your course data in a git repository (or some other version control system) and push it to version.aalto.fi or some such system. git and relevant tools are all installed in the images.
- You know that you are linked as an instructor to a course if, when you spawn that course’s environment, you get the `/course` directory.
- You can now make a direct link that will spawn a notebook server, for example for a course with a slug of `testcourse`: `https://jupyter.cs.aalto.fi/hub/spawn?profile=testcourse`. If the user is already running a server, it will **not switch to the new course**. Expect some subtle confusion with this and plan for it.
- We have a test course which you can use as a sandbox for testing nbgrader and courses. No data here is private even after deleted, and data is not guaranteed to be persistent. Use only for testing. Use the general use notebook for writing and sharing your files (using git).
- The course environments are not captive: students can install whatever they want. Even if we try to stop them, they can use the general use images (which may get more software at any time) or download and re-upload the notebook files. Either way, autograding is done in the instructors environment, so if you want to limit the software that students can use, this must be done at the autograding stage or via other hacks.
  - 1) If you want to check that students have **not** used some particular Python modules, have an hidden test that they haven’t used the module, like: `tensorflow` not in sys.modules.
  - 2) autograde in an environment which does not have these extra packages. Really, #2 is the only true solution. See the information under https://github.com/AaltoSciComp/isolate-namespace for information on doing this.
  - In all cases, it is good practice to pre-import all modules the students are expected to be able to use and tell students that other modules should not be imported.
- Students should use you, not us, as the first point of contact for problems in the system. Please announce this to students. Forward relevant problems to us.
- You can access your course data via SMB mounting at the URLs `smb://jhnas.org.aalto.fi/course/$courseslug/files/` and the course data using `smb://jhnas.org.aalto.fi/course/$courseslug/data/` (with Windows, use `\` instead of `/` and don’t include `smb://`). This can be very nice for managing files. This may mess up group-writeability permissions. It will take up to half a day to be able to access the course files after your request your course.
- You are the data controller of any assignments which students submit. We do not access these assignments on your behalf, and a submission of an assignment is an agreement between you and the student.
- You should always do random checks of a fair fraction of notebooks, to avoid unexpected problems.
- You can tell what image you have using `echo $JUPYTER_IMAGE_SPEC`.
- A notebook can tell if it is in the hub environment if the `AALTO_JUPYTERHUB` environment variable is set.
• You can install an identical version of nbgrader as we have using:

```
pip install git+https://github.com/AaltoSciComp/nbgrader@live
```

This may be useful if you get metadata mismatch errors between your system and ours. There used to be more differences, these days the differences are minimal because most of our important changes have been accepted upstream.

• You can get an environment.yml file of currently installed packages using:

```
conda env export -n base --no-builds
```

But note this is everything installed: you should remove everything from this file except what your assignments actually depend on, since being less strict will increase the chances that it’s reproducible. nbgrader should be removed (it pins to an unreleased development version which isn’t available), and perhaps the prefix should too. For actual versions installed, see base and standard dockerfiles in the singleuser-image repo.

FAQ

• Something with nbgrader is giving an error in the web browser. Try running the equivalent command from the command line. That will usually give you more debugging information, and may tell you what is going wrong.

• I see Server not running ... Would you like to restart it? This particular error also happens if there are temporary network problems (even a few seconds and it comes back). It doesn’t necessarily mean that your server isn’t running, but there is no way to recover. I always tell people: if you see this message, refresh the page. If the server is still running, it recovers. If it’s actually not running, it will give you the option to restart it again. If there are still network problems, you’ll see an error message saying that.

More info

• The Noteable is a commercial service using nbgrader and has some good documentation: https://noteable.edina.ac.uk/documentation/

Contact: CS-IT via the guru alias guru @ cs dot aalto.fi (students, contact your course instructors first).

For source code and reporting issues, see the main jupyterhub page.

See the separate instructors guide. This service may be either used as general light computing for your students, or using nbgrader to release and collect assignments.

Privacy policy

This system is managed by Aalto CS-IT. We do not store separate accounts or user data beyond a minimal database of usernames and technical logs of notebooks which are periodically removed (this is separate from your data). Your actual data is yours only and you are responsible for it. We do not access your data, but when necessary for the operation of the system, but we use and may look at file metadata such as permissions, timestamp, filename (stat filename). Your /notebooks directory may be deleted once it has been inactive for one year, and at the latest once your Aalto home directory is removed (after your account expires). Some courses will use the feedback/directory to return assignments to you.

The use of your own data and submission of data to your course instructors is the responsibility of you and the instructors.

See the separate privacy policy document for longer, less useful information.
FAQ and bugs

• I started the wrong environment and can’t get back to the course selection list. In JupyterLab, use the menu bar, “Hub->Control Panel”. On the classic notebooks, use the “Control panel” button on the top right. (Emergency backup: you can always change the URL path to /hub/home).

• Is JupyterLab available? Yes, and it’s nice. There are two general use instances that are actually the same, the only difference is one starts JupyterLab by default and one starts classic notebooks by default. Course environments always use classic notebooks, because the nbgrader assignment list only works there. To switch back and forth in any notebook server, change /tree in the URL to /lab/tree. If you want to use JupyterLab with a course’s files, first start that course’s server, get the assignments, then change to JupyterLab (change the URL, or stop and restart your server).

• Can I login with a shell? Run a new terminal within the notebook interface.

• Can I request more software be installed? Yes, let us know and we will include it if it is easy. We aim to have featureful environments by default, but won’t go so far as to install large specialist software. It should be in standard repositories (conda or pip for Python stuff).

• Can I do stuff with my class’s assignments and not have it submitted? You have your personal storage space /notebooks/, which you can use for whatever you want. You can always make a copy of the assignment files there and play around with them as much as you want - even after the course is over, of course.

• Are there other programming languages available? Currently there is Python, R, and Julia. More could be added if there is a good Jupyter kernel for it.

• What can I use this for? Intended uses include anything related to courses, own exploration of programming, own data analysis, and so on (see Terms of Use above). Long-term background processing isn’t good (but it’s OK to leave small stuff running, close the tab, and come back).

• When using nbgrader, how do I know what assignments I have already submitted? Currently you can’t beyond what is shown there.

• Can I know right away what my score is after I submit an assignment with nbgrader? nbgrader is not currently designed for this.

• Are there backups of data? Data storage is provided by the Aalto Teamwork system. There are snapshots available in .snapshot in every directory (you have to ls this directory in a shell using its full name for it to appear the first time). This service is not designed for long term data storage, and you should back up anything important because it will be lost after about one year or when your Aalto account expires. You should use git as your primary backup mechanism, obviously.

• Is git installed? Yes, and you should use it. Currently you have to configure your username and email each time you use it, because this isn’t persistent (because home directories are not persistent). Git will guide you through doing this. In the future, your Aalto directory name/email will be automatically set. As a workaround, run git config without the --global option in each repository.

• I don’t see “Assignment list”. You have probably launched the general use server instead of a course server. Stop your server and go spawn the notebook server of your course.

• I’m getting an error code Here are the ones we know about:
  – 504 Gateway error: The hub isn’t running in background. This may be hub just restarting or us doing maintenance. If it persists for more than 30 minutes, let someone know.

• Stan/pystan/Rstan don’t work. Stan needs to do a memory-intensive compilation when your program is run. We can’t increase our memory limits too much, but we have a workaround: you need to tell your program to use the clang compiler instead of the gcc compiler by setting the environment variables CC=clang and CXX=clang++. For R notebooks, this should be done for you. For RStudio, we don’t know. For Python, put the following in your notebook:
import os
os.environ['CC'] = "clang"
os.environ['CXX'] = "clang++"

We should set this the default, but want to be sure there are no problems first.

- **RStudio doesn’t appear.** It seems that it doesn’t work from the Edge browser. We don’t know why, but try another browser.

- **I’ve exceeded my quota.** You should reduce the space you use, the quota is 1GB. If this isn’t enough and you actually need more for your classes, tell your instructor to contact us. To find large directories: open a terminal and run `du -h /notebooks/ | sort -h` to find all large files. Then clean up that stuff somehow, for example `rm -r`. Note that `.home/.local/share/jupyter/nbgrader_cache` will continue to grow and eventually needs to be cleaned up - after the respective course is done.

- **I don’t see the assignments for my course.** There are different profiles you can start, and you can’t tell which profile you have started. Go back to the hub control panel and restart your server. To be more precise, click the “Control Panel” in the upper-right corner, then click “Stop my Server”, wait a little bit, then click “Start My Server” and choose the profile for your course.

### More info

Students, your first point of contact for course-related matters and bugs with JupyterHub should be your instructors, not us. They will answer questions and send the relevant ones to us. But, if you can actively help with other things, feel free to comment via Github repositories below.

The preferred way to send feedback and development requests is via Github issues and pull requests. However, we’re not saying it’s a bad idea to give Github all our information, so you can also send tickets to CS-IT.

Students and others who have difficulty in usage outside of a course can contact CS-IT via the *guru* alias.

Notebooks are not an end-all solution: for an entertaining look at some problems, see “I don’t like notebooks” by Joel Grus or less humorous pitfalls of Jupyter notebooks. Most of these aren’t actually specific to notebooks and JupyterLab makes some of the problems better, but thinking hard about the downsides of notebooks makes your work better no matter what you do.

Our source is open and on Github:

- **single-user image** (everything about a user’s environment)
- **server itself** (logging in, course profiles, etc).

### 4.1.10 Remote Jupyter Notebook on shell servers

See also:

We now have a General use student/teaching JupyterHub installation which may serve your uses more simply.

Here we describe how you can utilise Aalto computing resources for Jupyter Notebook remotely. The guide is targeted for UNIX users at the moment.

Aalto provides two “light computing” servers: `brute.org.aalto.fi`, `force.org.aalto.fi`. We demonstrate how to launch a Jupyter Notebook on `brute` and access it on your laptop.

```
ssh username@brute.org.aalto.fi

# Create your Kerberos ticket
kinit
```

(continues on next page)
Fig. 2: <System activity on Brute>
# Create a session. I use tmux

tmux

# Load Anaconda
module load anaconda

# Create your env
conda create -n env-name python=3.6 jupyter

# Activate your python environment
source activate env-name

# Launch jupyter notebook in headless mode and a random port number
jupyter notebook --no-browser --port=12520

Note: You might get messages like The port 12520 is already in use, trying another port while starting the notebook server. In that case, take note of the port the server is running in, e.g.:

```
```

and replace “12520” below with the correct port number, 12470 in this case.

Now back to your laptop

```
# Forward the port
ssh -L 12520:localhost:12520 -N -f -l username brute.org.aalto.fi
```

Now launch your browser and go to http://localhost:12520 with your token.

## 4.1.11 Paniikki: Computer Lab For Students

Paniikki is a cutting edge computer lab in the computer science department. It is located in T-building C106 (right under lecture hall T1). This documentation is a Paniikki cheatsheet.

For more services directed at students, see Welcome, students!.

### The name

Paniikki means “panic” in English which is a fascinating name as people in panic are in panic. I don’t know which comes first, the space or the emotion. Anyway, people experience the both simultaneously.
Fig. 3: The blue box at the entrance is Paniikki
Access

Physical

You can access Paniikki in the T-building C106. It is right by the building’s main entrance (you can see it through the windows by the building’s main entrance).

Remote

You can ssh via the normal Aalto shell servers kosh and lyta. Going through them, you can then ssh to one of the Paniikki computers. Be warned, there is no guarantee that you get an empty one . . . if it seems loaded (use htop to check), try a different one.

You can find the hostnames of the Paniikki computers on aalto.fi.

Hardware

<table>
<thead>
<tr>
<th>CPU properties</th>
<th>Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz</td>
</tr>
<tr>
<td>Architecture</td>
<td>x86_64</td>
</tr>
<tr>
<td>CPU(s)</td>
<td>12</td>
</tr>
<tr>
<td>Thread(s) per core</td>
<td>2</td>
</tr>
<tr>
<td>max MHz</td>
<td>4000.0000</td>
</tr>
<tr>
<td>Virtualization</td>
<td>VT-x</td>
</tr>
<tr>
<td>L1d cache</td>
<td>32K</td>
</tr>
<tr>
<td>L1i cache</td>
<td>32K</td>
</tr>
<tr>
<td>L2 cache</td>
<td>256K</td>
</tr>
<tr>
<td>L3 cache</td>
<td>15360K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPU properties</th>
<th>Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>NVIDIA Quadro P5000</td>
</tr>
<tr>
<td>Core</td>
<td>GP104GL (Pascal-based)</td>
</tr>
<tr>
<td>Core clock</td>
<td>1607 MHz</td>
</tr>
<tr>
<td>Memory clock</td>
<td>1251 MHz</td>
</tr>
<tr>
<td>Memory size</td>
<td>16384 MiB</td>
</tr>
<tr>
<td>Memory type</td>
<td>256-bit GDDR5X</td>
</tr>
<tr>
<td>Memory bandwidth</td>
<td>320</td>
</tr>
<tr>
<td>CUDA cores</td>
<td>2560</td>
</tr>
<tr>
<td>CUDA compute capability</td>
<td>6.1</td>
</tr>
<tr>
<td>OpenGL</td>
<td>4.5</td>
</tr>
<tr>
<td>OpenCL</td>
<td>1.2</td>
</tr>
<tr>
<td>Near GeForce Model</td>
<td>GeForce GTX 1080</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory properties</th>
<th>Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>32GiB</td>
</tr>
</tbody>
</table>
Software

First thing first, you don’t have sudo rights on Aalto classroom machines and you can’t, because they are shared. We provide the most used SW and if you need more you could inquire via servicedesk@aalto.fi. We try to have a good base software that covers most people’s needs.

<table>
<thead>
<tr>
<th>What?</th>
<th>How?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python via Anaconda</td>
<td>module load anaconda</td>
</tr>
<tr>
<td>Python (system)</td>
<td>Default available</td>
</tr>
<tr>
<td>Tensorflow</td>
<td>in the Python environments, e.g. anaconda above</td>
</tr>
</tbody>
</table>

Modules

In short, module is a software environment management tool. With module you can manage multiple versions of software easily. Here are some sample commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module load NAME</td>
<td>load module</td>
</tr>
<tr>
<td>module avail</td>
<td>list all modules</td>
</tr>
<tr>
<td>module spider NAME</td>
<td>search modules</td>
</tr>
<tr>
<td>module list NAME</td>
<td>list currently loaded modules</td>
</tr>
<tr>
<td>module show NAME</td>
<td>details on a module</td>
</tr>
<tr>
<td>module help NAME</td>
<td>details on a module</td>
</tr>
<tr>
<td>module unload NAME</td>
<td>unload a module</td>
</tr>
<tr>
<td>module save ALIAS</td>
<td>save module collection to this alias (saved in ~/.lmod.d/)</td>
</tr>
<tr>
<td>module restore ALIAS</td>
<td>load saved module collection (faster than loading individually)</td>
</tr>
<tr>
<td>module purge</td>
<td>unload all loaded modules (faster than unloading individually)</td>
</tr>
</tbody>
</table>

There are some modules set up specifically for different courses: if you just load the environment with "module load", you will have everything you need.

Read the details in Module environment page.

Example 1

Assume we are in Paniikki and wants to do our homework for CS-E4820 Machine Learning: Advanced probabilistic methods. In the course students use Tensorflow and Edward.

```bash
# Check available modules
$ module load courses/  # Tab to auto-complete

# Finally you will complete this
$ module load courses/CS-E4820-advanced-probabilistic-methods.lua

# Check the module you loaded
$ module list

Currently Loaded Modules:
    1) courses/CS-E4820-advanced-probabilistic-methods
```
# Check the packages
$ conda list    # You will see Tensorflow and etc.

# Launch Jupyter
$ jupyter notebook

# Do your homework

# You are done and want to un-load all the modules?
$ module purge

### Example 2: General Python software

Need Python and general software? The anaconda modules have Python, a bunch of useful scientific and data packages, and machine learning libraries.

```bash
# Latest Python 3
$ module load anaconda

# Old Python 2
$ module load anaconda2
```

### Example 3: List all software

You can check all other modules as well

```bash
$ module avail
```

You want to use Matlab?

```bash
$ module load matlab/2017b
$ matlab
```

### Questions?

If you have any question please contact servicedesk@aalto.fi and clearly mention the Paniikki classroom in the message.

### 4.1.12 HTCondor

**Note:**
- SCIP courses: look for Introduction to distributed computing with HTCondor
HTCondor (formerly known as just Condor) is a computing scheduler developed at University of Wisconsin-Madison. This allows users to run their binaries on Aalto Linux workstations without explicit logging to desktop machines. Condor takes care of choosing the right workstation, setting correct job priority and taking care of cleaning the output. Condor distributes, schedules, executes and returns the result. So handmade farming is not needed.

HTCondor status at Aalto and support

Condor installations are department specific. Here is a list of departments that have HTCondor software installed on their Ubuntu workstations.

<table>
<thead>
<tr>
<th>Department / school</th>
<th>Support contact</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHYS &amp; NBE / SCI</td>
<td>Aalto IT servicedesk *</td>
<td>joint installation, installed on all the Ubuntu workstations</td>
</tr>
<tr>
<td>CS / SCI</td>
<td>Aalto IT servicedesk *</td>
<td>installed on all the Ubuntu workstations</td>
</tr>
<tr>
<td>MATH / SCI</td>
<td>Matti Harjula and Kenrick Bingham</td>
<td>installed on about 50 newer Ubuntu workstations</td>
</tr>
</tbody>
</table>

The instructions below are common to all the departments if not mentioned otherwise.

* Getting help: your department IT guys have responsibility over the HTCondor installation. Best way to reach them is to drop an email to the Aalto IT servicedesk including info like: your department, Linux workstation name and type of problem.
HTCondor official manuals

The detailed manual can be found from https://research.cs.wisc.edu/htcondor/manual/. Current version of Condor we have can be checked with `condor_q -version`.

Before you run with Condor

It is recommended that you compile you binary statically. If you have used shared libs (or you get from someone code that has not been compiled statically), make sure that you set your environment correctly and use `getenv = true` option in Condor submit script.

No large MPI jobs (over the net) are allowed with Condor. For any large MPI or multithread job, please either run on your local workstation only or on other resources like Triton.

Condor is well suited for short time serial runs (like overnight), or for small (2-4 CPUs) parallel runs that can be run within one machine. Long runs (over 12 hours) are possible, but remember that Condor runs on local workstations, and uses only idle CPU cycles, i.e. some currently unused workstation during the day and all of them during night. Local usage is of higher priority and thus submitted Condor job that hurts local user will be suspended.

Always use `should_transfer_files = yes` in your Condor submit script. This way you make sure that all IOs will go to local directory assigned to HTCondor on a local worker instead of shared NFS (be it /home or alike).

Run your code with Condor

- Discover condor pool status with `condor_status` or with `condor_status -available` to find out which machines are available for jobs. This step is to make sure that condor pool is available.
- Compile a statically linked binary.
- Create a condor submission script, like `job_.cond` below
- Submit the job to condor pool with `condor_submit job_.cond`
- Manage your job(s) with `condor_q`, `condor_rm`

It may take several minutes for code to start running. Check out `condor.log` for any useful log information.

Job script examples

CS users should use `universe = local`

```bash
# job_1.cond -- ready to run serial code example

executable = serial.bin
universe = vanilla
output = serial.out
error = serial.err
log = condor.log
should_transfer_files = YES
queue
```

```bash
# job_2.cond -- Condor serial job submission script example

# define job specific vars to be used later in this script
# this should be an absolute path, or path from current working dir
DIR=myrun
```

(continues on next page)
# setting up base directory for input, output, error and log files, executable path
initialdir = $(DIR)

# Define executable to run, it can be arch specific, or just some generic code
executable = mycode

# memory requirements, if any
request_memory = 512 MB

# Condor universe. Default Vanilla, others haven't been configured/tested
universe = vanilla

# the file name specified with 'input' should contain any keyboard input the program
# requires
input = input.txt

# note, that command-line arguments are specified by the 'arguments' command below
#output = input.txt

# and output files
# note, that input, output, log and error files will/should be in 'initialdir'
output = $(cluster).out

# Errors, if any, will go here
error = $(cluster).err

# Always define log file, so that you know what happened to your job(s)
log = condor.log

# email for job notifications, when it is completed or finished with errors
#notify_user = firstname.lastname@aalto.fi

# Additional environment vars
#environment = "PATH=$ENV(PATH):/home/user/bin"

# replicate your current working environment on the worker node
# useful when you have some specific vars like PATH, LD_LIBRARY_PATH or other defined
# with 'module'
getenv = true

# code arguments, if any
#arguments = -c cmd_input.conf

# Trasferring your files to a system the job is going to run on
# that is the recommended method, to avoid NFS traffic
should_transfer_files = yes
transfer_input_files = cmd_input.conf,input.txt
when_to_transfer_output = ON_EXIT_OR_EVICT

# Some specific requirements, if any. By default Condor will run job on a machine
# which has
# the same architecture and operating system family as the machine from which it was
# submitted.
# Here is we want the worker node would be Ubuntu 12.04 with 4 CPU cores or more
#requirements = (OpSysLongName >= "Ubuntu 12.04") && (TotalCPus >= 4)
Condor commands

- `condor_q -analyze <condor_job_id>` # your running/pending jobs diagnostics (for all your jobs at once if `job_id` is missing)
- `condor_q -global` # list all/everyone’s jobs at pool
- `condor_q -version` # find out installed condor version
- `condor_status -available` # list available computers for your job
- `condor_status -state -total` # Condor pool resources in total
- `condor_status HOSTNAME` # show status for a specific host (HOSTNAME.hut.fi in this case), where number of slots gives number of CPU cores available
- `condor_status -long vesku` # show all details for a specific host
- `condor_status -constraint 'OpSysLongName>="Ubuntu 12.04"'` # list Ubuntu 12.04 workstations only
- `condor_rm <condor_job_id>` # remove particular job
- `condor_rm -all` # remove all user jobs
- `condor_rm -constraint 'JobStatus =!= 2'` # remove all user jobs that are not currently running
- `condor_hold <job_id>` # hold your Condor job(s) in the queue
- `condor_release <job_id>` # release job(s) previously holded in the queue
- `(NOTE: doesn’t work on Ubuntu, so anywhere at Aalto) condor_compile [cc \ | f77 \ | g++ \ | make \ | ...]` # relink an executable for checkpointing with Standard universe; not installed on Ubuntu 12.04, see Checkpointing section below
- `condor_history` # list the completed jobs submitted from the workstation you run this command on

Startup script requirements= can be always tested with `condor_status -constraint`. Like in the above `job_2.cond` example:

- `condor_status -constraint 'OpSysLongName>="Ubuntu 12.04"' && (TotalCPus >= 4)'` -available

More commands and their usage examples you can find at [Condor User Manual](https://www.cs.wisc.edu/~skiena/condor/).

Additional “requirements”/“constraints” options that have been configured on PHYS workstations only: CPUModel, CPUModelName, TotalFreeMemory. The later one in MB, reports currently available free memory according to `/proc/meminfo`. Can be useful for large memory jobs, see example below.

```bash
# ask for machine with more than 4GB of free memory
requirements = (TotalFreeMemory >= 4000)
```
Checkpointing and condor_compile

HTCondor has no checkpointing or remote system calls support on Ubuntu (according to manual pages).

HTCondor config

Machine in considered to be free if: no user activity within 15 min (keyboard or mouse), average load < 30%, and no condor job already running.

Running job will be suspended if: local workstation user became active (on hold) or CPU busy for more than 2 min and job has been running more than 90 sec.

Suspended job will be resumed if: machine has been free for 5 min.

Suspended job is killed if: it has been suspended for 4 hours (Vanilla universe) or hasn’t completed checkpointing within 10 min (Standard universe) or higher priority job is waiting in the queue.

Job will be preempted if: it uses more memory than available for its slot (killed and send back to queue).

FAQ

Condor has support on running jobs under shared filesystem. Should I use this?

This is a bad idea. Keep using Condor’s default local directory (somewhere on the local harddrive, department specific settings), otherwise, several jobs using NFS constantly (either home or any other remotely mounted) would make it really slow. Use

```
should_transfer_files = YES
transfer_input_files = file1.dat,file2.txt
```

options instead. Then condor will copy all required (specified) files to its local spool directory and run jobs locally. Only when finished, it will return files back to the original submitting directory. This original submitting directory should not be a NFS mounted directory such as your home directory, as in the Aalto environment those are mounted with Kerberos security, and if the Kerberos ticket has expired because you aren’t working on your workstations, condor will not be able to access this directory and your job results will be lost.

My job is in ‘Idle’ state, while there are resources available

Job may take several minutes to start, if it takes longer, check out job log (defined with log = directive in the submit script) and then run condor_q -analyze <job_id> to see possible reasons. More debugging options at condor_q manual.

I’ve copy/pasted example files from this page, but when try to run they produce some errors

Should be this wiki specific. Noticed (with cat -A filename) that copy/pasted text includes bunch of non-ascii characters.

Got it fixed with perl -pi -e 's/[[[:ascii:]]] //g' filename
Aalto scientific computing guide

Additional files/scripts

Files that may be useful with condor:

- cg – A script that works as condor_q but also prints the executing host

```perl
#!/usr/bin/perl

use POSIX;

$user=$ENV{'LOGNAME'};
$now=`date +%s`; $now=$now/\n/;
$str=" -cputime -submitter $user ";
for $i (0..$#ARGV) {
  $str.= $ARGV[$i-1];
}
if($ARGV[0] eq "all") {$str=" -global -cputime -currentrun";)
if($ARGV[0] eq "j") {system("condor_q -global -cputime -currentrun -submitter $user|egrep '(jobs|Schedd)'"};exit(0);
if($ARGV[0] eq "rm") {$str=`condor_q -submitter $user -format "%d\n\nClusterId\n\n`|xargs`;print "condor_rm $str";exit(0);
foreach(`condor_q -long $str`) {
  s/\n/; s/\n/g;
  if(m/^Iwd\s*=\s*(\S+)/) { $iwd=$1; }
  if(m/^RemoteHost\s*=\s*(\S+)/) { $rh=$1; }
  if(m/ServerTime/) {
    $iwd=~/Iwd\s*=\s*(\S+)/; push(@iwds, "$rh	 $iwd");
  }
}
foreach(`condor_q $str`) {
  s/\n/; if(/\s+\d+\.\d/) {
    $iwd=shift(@iwds);
    $_.=" 
    print "$_.\n";
  }
sub runtime() {
  my($now, $st)=@_; $str=localtime($now-$st-7200);
  $str=\n/g;
  $str=\n/g;
  $str=\n+/g;
  split(/,\$str); $d=$_[2]-1; $t=$_[3];
```
if($d>0) {$ret="$d+$t";}else{$ret=$t;}
return $ret;
}

- turbomole.cond, run_ridft510_condor.scr– pair of scripts for running TurboMole or AMBER (thanks to Markus Kaukonen)

```bash
#!/bin/sh
source /etc/profile
source /etc/bashrc
source /etc/profile.d/fyslab-env.sh

AMBERHOME=${HOME}/bin/Amber10
TURBODIR=${HOME}/bin/Turbo5.10/

PATH=$PATH:$TURBODIR/scripts
PATH=$PATH:$TURBODIR/bin/`sysname`

export PATH
export PATH="${AMBERHOME}/exe:${AMBERHOME}/bin:${PATH}"
export PATH="${HOME}/bin:${PATH}"

ulimit -s unlimited
#ulimit -a > mylimits.out

jobex -ri -c 200 > jobex.out
```
4.1.13 Aalto Gitlab

https://version.aalto.fi is a Gitlab installation for the Aalto community. Gitlab is a git server and hosting facility (an open source Github, basically).

Note:
- This page is about https://version.aalto.fi, the Aalto gitlab installation.
- scicomp/git contains our pointers for Git usage in general.
- Git migration contains information on switching from subversion or other git repositories to Gitlab.

Git in general

Git seems to have become the most popular and supported version control system, even if it does have some rough corners. See the general git page on this site for pointers.

Aalto Gitlab service

Aalto has a self-hosted Gitlab installation at https://version.aalto.fi, which has replaced most department-specific Gitlabs. With Aalto Gitlab, you can:
- Have unlimited private repositories
- Have whatever groups you need
- Get local support

The Aalto instructions can be found here, and general gitlab help here.

All support is provided by Aalto ITS. Since all data is stored within Aalto and is managed by Aalto, this is suitable for materials up to the “confidential” level.

Extra instructions for Aalto Gitlab

Always login with HAKA wherever you see the button. To use your Aalto account otherwise, use username@aalto.fi and your Aalto password (for example, use this with https pushing and pulling). But, you really should try to configure ssh keys for pushing and pulling.

For outside/public sharing read-only, you can make repositories public.

If you need to share with an outside collaborator, this is supported. These outside partners can access repositories shared with them, but not make new ones. They will get a special gitlab username/password, and should use that with the normal gitlab login boxes. To request an collaborator account, their Aalto sponsor should go here to the request form (employees only). (You can always set a repository as public, so anyone can clone. Another hackish method is to add ssh deploy keys (read-only or read-write) for outside collaborators, but this wouldn’t be recommended for serious cases.)

For public projects where you want to build a community, you can also consider Github. There’s nothing wrong with having both sites for your group, just make sure people know about both. Gitlab can have public projects, and Github can also have group organizations.

NOTE! If your work contract type changes (e.g. staff -> visitor, student->employee, different department), the Aalto Version blocks the access as a “security” measure. Please contact Aalto ITS Servicedesk <servicedesk@aalto.fi> to unblock you. This is annoying, but can’t be fixed yet.
The service doesn’t have quotas right now, but has limited resources and we expect everyone to use disk space responsibly. If you use too much space, you will be contacted. Just do your best to use the service well, and the admins will work with you to get your work done.

**CodeRefinery Gitlab and Gitlab CI service**

CodeRefinery is a publicly funded project (by Nordforsk / Nordic e-Infrastructure Collaboration) which provides teaching and a GitLab platform for Nordic researchers. This is functionally the same as the Aalto Gitlab and may be more useful if you have cross-university collaboration, but requires more activation to set up.

They also have a Gitlab CI (continuous integration) service which can be used for automated building and testing. This is also free for Nordic researchers, and can be used even with Aalto Gitlab. Check their repository site info, if info isn’t there yet, then mail their support asking about it.

**Recommendations**

version.aalto.fi is a great resource for research groups. Research groups should create a “Gitlab group” and give all their members access to it. This way, code and important data will last longer than single person’s time at Aalto. Add everyone as a member to this group so that everyone can easily find code.

Think about the long term. Will you need access to this code in 5 years, and if so what will you do?

- If you are a research group, put your code in a Gitlab group. The users can constantly switch, but the code will stay with the group.
- If you are an individual, plan on needing a different location once you leave Aalto. If your code can become group code, include it in the group repository so at least someone will keep it at Aalto.
- Zenodo is a long-term data archive. When you publish projects, consider archiving your code there. (It has integration with Github, which you might prefer to use if you are actually making your code open.) Your code is then citable with a DOI.
- In all cases, if multiple people are working on something, think about licenses at the beginning. If you don’t, you may be blocked from using your own work.

**FAQ**

- **What password should I use?** It is best to use HAKA to log in to gitlab, in which case you don’t need a separate gitlab password. To push, it is best to use ssh keys.
- **My account is blocked!** That’s not a question, but Gitlab blocks users when your Aalto unit changes. This is unfortunately part of gitlab and hasn’t been worked around yet. Mail servicedesk@aalto.fi with your username and request “my version.aalto.fi username XXX be unblocked (because my aalto unit changed)” and they should do it.
- **What happens when I leave, can I still access my stuff?** Aalto can only support it’s community, so your projects should be owned by a group which you can continue collaborating after you leave (note that this is a major reason for group-based access control!). Email servicedesk for information on what to do to become an external collaborator.
- **When are accounts/data deleted?** The deletion policy is findable in the privacy policy. In 2017, it’s 6 months after Aalto account closed, 24 months after last login, or 12 months after last login of an external collaborator.
- **Are there continuous integration (CI) services available?** Not from Aalto, but the CodeRefinery project has free CI services to Nordics, see their site and the description above.
4.1.14 Python

The scientific python ecosystem is also available on Aalto Linux workstations, including the anaconda (Python 3) and anaconda2 (Python 2) modules providing the Anaconda python distribution. For a more indepth description see the generic python page under scientific computing docs.

The “neuroimaging” environment

On the Aalto Linux workstations and Triton, there is a conda environment which contains an extensive collection of Python packages for the analysis of neuroimaging data, such as fMRI, EEG and MEG.

To use it on Aalto Ubuntu workstations and VDI:

```
$ ml purge
$ ml anaconda3
$ source activate neuroimaging
```

To use it on Triton:

```
$ ml purge
$ ml neuroimaging
```

To see the full list of packages what are installed in the environment, use:

```
$ conda list
```

Some highlights include:

- Basic scientific stack
  - numpy
  - scipy
  - matplotlib
  - pandas
  - statsmodels
- fMRI:
  - nibabel
  - nilearn
  - nitime
  - pysurfer
- EEG/MEG:
  - mne
  - pysurfer
- Machine learning:
  - scikit-learn
  - tensorflow
  - pytorch
• R:
  – rpy2 (bridge between Python and R)
  – tidyverse

Finally, if you get binaries from the wrong environment (check with `which BINARYNAME`), you may need to update the mappings with:

```
$ rehash
```

**MNE Analyze**

Note: this was tested only for NBE workstations. If you wish to run `mne_analyze` from your workstation you should follow this procedure. Open a new terminal and make sure you have the `bash` shell (`echo $SHELL`, if you do not have it, just type `bash`) and then:

```
$ module load mne
$ source /work/modules/Ubuntu/14.04/amd64/common/mne/MNE-2.7.4-3434-Linux-x86_64/bin/
    → mne_setup_sh
$ export SUBJECTS_DIR=PATHTOSUBJECTFOLDER
$ export SUBJECT=SUBJECTID
$ mne_analyze
```

Please note that the path of the “source” command might change with most up to date versions of the tool. Please note that the “PATHTOSUBJECTFOLDER” and “SUBJECTID” are specific to the data you have. Please refer to MNE documentation for more help on these.

**Mayavi**

If you experience problems with the 3D visualizations that use Mayavi (for example MNE-Python’s brain plots), you can try forcing the graphics backend to Qt5:

• For the Spyder IDE, set Tools -> Preferences -> Ipython console -> Graphics -> Backend: Qt5

• For the ipython consoles, append `c.InteractiveShellApp.matplotlib = 'qt5'` to the `ipython_config.py` and `ipython_kernel_config.py` configuration files. By default, these can be found in `~/.ipython/profile/default/`.

• In Jupyter notebooks, execute the magic command `%matplotlib qt5` at the beginning of your notebook.

**Installation of additional packages**

The “neuroimaging” environment aims to provide everything you need for the analysis of neuroimaging data. If you feel a package is missing that may be useful for others as well, contact Marijn van Vliet. To quickly install a package in your home folder, use `pip install <package-name> --user`. 

---

**4.1. Aalto tools**
4.1.15 Open Source at Aalto

Note: This policy was developed at the Department of Computer Science, in conjunction with experts from Research and Innovation services (both the legal and commercialization sides) with the intention of serving the wider community.

After more research, we have learned that this policy is, in fact, de-facto applicable to all of Aalto, it is just extremely unclear that open source is actually allowed. Thus, this policy can be seen as best practices for all of Aalto. However, everyone (including CS) has more rights: one does not have to use this policy. You don’t have to use an open source license. IP ownership may be in more limited hands, so that you need fewer agreements to release.

However, we strongly encourage you to use this policy anyway. If you use this, you know that you are safe and have all permissions to make open source, regardless of your particular funding situation. It also ensures that you make proper open source software, for maximum benefit and open science impact.

References at bottom.

Researchers make at least three primary outputs: publications, software, and data. This policy aims to make openly releasing all types of work as straightforward the traditional academic publishing process.

This document describes the procedure for Aalto employees releasing the output of their work openly (open source software, data, and publications). Aalto University encourages openness. This policy covers only cases where work can clearly be released openly with no bureaucracy needed. It does not cover complex cases, such as commercial software, work related to inventions, complex partnership agreements, etc. The policy is voluntary, and provides a right to release openly, but does not require it or preclude any other university process. (Thus it’s more of a guideline than a policy.) It only is relevant when the creator has an employment relationship with Aalto. If they don’t (e.g. students), they own their own work unless there is some other agreement in place (e.g. their own funding contract, grant, etc). Still, they can use this same process with no extra bureaucracy needed.

We realize that this policy does not cover all cases. We aim to cover the 99% case, and existing processes are used for complicated cases. Aalto Innovation Services provides advice on both commercialization and open source release.

This policy is for public licensing only (one to many). You must go through Research and Innovation Services for anything involving a multi-party agreement.

Why release?

The more people who see and build on our work, the more impact we can have. If this isn’t enough, you get more citations and attention. While we can’t require anything, we strongly encourage that all work is either made open source or taken through the commercialization process. If you don’t know what to do, don’t worry: they are not mutually exclusive. Proper open-source licensing can protect your ability to commercialize later. Talk to Innovation Services. They like open source, too, and will help you find the right balance. Anyway, if work matches the criteria in this policy, it probably has limited commercial potential anyway: what is more important is your own knowledge and skills that went into it.

You want to add a proper open source license to your work, rather than just putting code on some webpage. Without a license, others can not build on your code, making your impact limited. No one will build on you, and eventually your work rots and gets lost.

You always want to go through this process as soon as possible at the beginning of a project: if you don’t, it becomes much harder to track everyone down.

You shouldn’t release as open source (yet) if your work is intentionally commercial or contains patentable inventions. In these cases, contact Innovation Services. In the second case (patentable inventions), according to Finnish legislation you are actually required to report the invention to Innovation Services.
Traps and acting early

Intellectual property rights don’t give you the right to do anything - they give you the right to block others from doing something. Thus, it is very important that you don’t end up in a situation where others can block you, and that means thinking early.

Decide on a license as soon as possible. Once it goes into the repository, future contributors implicitly agree to it. Otherwise, you are stuck trying to find all past contributors and get their agreement.

Another common trap is non-open source friendly grants. Not many outright ban it, but some require permission from all partners, and if there are a lot then this becomes close to impossible. Ask in advance, but in the worst case it might be you just can’t write software at the times you are paid by these projects!

Step-by-step guide for release under this policy

1. Do these steps at the beginning of your project, not at the end!
2. Check if the work is covered under the “conditions for limited commercial potential” in the policy.
3. Choose a proper license to match your needs. See below for information. It must be open source, and you can not transfer any type of exclusive license away - Aalto keeps full right to future use.
4. Get the consent of all authors and their supervisors and/or funders. There are no particular requirements for this, the only need is proving it later in case a question ever arises. You should also make sure that your particular funding source/collaboration agreements don’t have any further requirements on you. (For example, some grant agreements may say no GPL-type licenses without consent of all partners.) Your advisor (and Research and Innovation Services) can help you with this.
   - If you are funded by Aalto basic funding, you by default have permission. Same goes for other big public funding agencies (Academy, EU… but the grant can always override this).
   - If you are in services, follow your source of funding. At the very worst, whoever is responsible for your funding can decide, but it may be someone lower too.
5. You are responsible for making sure that you have the right to release your code. For example, that there are no other agreements other rights (intellectual property and privacy), legal restrictions, or anything else restricting a release. Also, any other included software must have compatible licenses.
6. Put a copyright license in the source repository. In the best case, each individual source file should list copyright and authors, but in practice if you don’t do this it’s not too much of a problem. Make sure that the license disclaims any warranty (almost all licenses will do this). After this, contributors implicitly consent to the license. If you have an important case, ask explicitly too. The important thing is that you have more evidence than the amount of scrutiny you might get (low in typical projects, will be higher if your project becomes more important).
7. This policy is seen as Aalto transferring the rights to you to release, not Aalto releasing itself (just the same as with publications). Release in your own name, but you can(+should) list your affiliations.
8. Make your code public if/when you want. No particular requirements here, but see below for best practices.

Any borderline or questionable cases should be handled by the existing innovation disclosure process.

In addition to the above requirements, the following are best practices:
1. You can’t require that people cite you, but you can ask nicely. Make it easy to do this! Include the proper citations directly in the README. Make your code itself also citeable by publishing it somewhere (Github, Zenodo, …).
2. Put on a good hosting location and encourage contributions. For example, Github is the most popular these days, but there are plenty of others. Welcome contributions and bug reports, and build on them. Make yourself the hub of expertise of your knowledge and methods.
Choosing a license

Under this policy, any Creative Commons, Open Source Initiative, and Free Software Foundation approved open source licenses are usable. However, you should not try to be creative, and use the most common license that serves your needs.

Top-level recommendations:

1. Use this nice site: https://choosealicense.com/. It contains everything you need to know, including what is here. If you need something more specific you can have a look at http://oss-watch.ac.uk/apps/licdiff/.

2. MIT for software which should be basically public domain, Apache 2.0 for larger almost-public domain things (the Apache license protects more against patent trolling). Anyone can use this for any purpose, including putting it in their own proprietary, non-open products.

3. GNU General Public License (GPL) (“v2 or any later version”) for software which you may try to commercialize in the future. This license says that others can not make it closed-source without your consent. Others can use it for commercial purposes, but all derivative work must also be made open source - so you keep an advantage.

For special cases:

1. Lesser GNU General Public License (LGPL, GPL with classpath exception) type licenses. Suitable where the GPL would be appropriate but the software is a library. It can be embedded within other proprietary products, but the code itself must stay open.

2. The Affero GPL/LGPL. These get around the “webservice loophole”: if your code is available via a webserver, the code running it must stay open.

3. CC-BY for other non-software output.

Discussion:

- Most public domain → MIT / Apache 2 > CC-BY > LGPL > GPL > AGPL → Most protection against proprietary use
- If you think you might want to commercialize in the future: ask innovation services and they’ll help you release as open source now and preserve commercialization possibilities for the future.

The policy

Open Source Policy

Covered work

1. Software

2. Publications and other writing (Note that especially in this case, it is common to sign away full rights. This is a case where you do more than this policy says.)

3. Data
Conditions for limited commercial potential

This policy supports the release of work with limited commercial potential. Work with commercial potential should be assessed via Aalto’s innovation process.

1. If work’s entire novelty is equally contained in academic publications, there is usually little commercial value. Examples: code implementing algorithms, data handling scripts.

2. Similarly, work which only is a byproduct of academic publications or other work probably has limited commercial value, unless some other factor overrides. For example: analysis codes, blog posts, datasets, other communications.

3. Small products with limited independent value. If the time required to reproduce the work is small (one week or less), there is likely not commercial value. For example: sysadmin scripts, analysis codes, etc. Think about the time for someone else to reproduce the work given what you are publishing, not the time it took for you to create it.

4. Should a work be contributing to an existing open project, there is probably little commercial value. For example: contribution to existing open-source software, Wikipedia edits, etc.

5. NOT INCLUDED: Should work contain patentable elements or have commercial potential, this policy does not apply and it should be evaluated according to the Aalto innovation process. Patentable discoveries are anything which is a truly new, non-obvious, useful inventions. In case of doubt, always contact Innovation Services! Indicators for this category: actually novel, non-obvious, useful, and actually an invention. Algorithms and math usually do not count, but expressions of these can.

6. NOT INCLUDED: Software designed for mass-market consumption or business-to-business use should be evaluated according to the Aalto innovation process. Indicators for this category: large amount of effort, software being a primary output.

Ownership of intellectual property rights at Aalto

1. This policy covers work of employees whose contracts assign copyright and other intellectual property rights of their work to Aalto. However, the Aalto rules for ownership of IP are extremely difficult, so see the last point.

2. Your rights are assigned to Aalto if you are funded by external funding, or if there are other Aalto agreements regarding your work.

3. If neither of the points in (2) apply to you AND your work is independent (self-decided and self-directed), then according to Finnish law you own all rights to your own work. You may release it how you please, and the rest of this policy does NOT apply (but we recommend reading it anyway for valuable advice). Aalto Innovation Services can serve you anyway.

4. Rather than figure out the the ownership of work, this policy is written to apply to all work, so that you do not need to worry about this.

Release criteria and process

1. This policy applies to copyright only, not other forms of intellectual property. Should a work contain other intellectual property (which would not be published academically), this policy does not apply. In particular, this policy does not cover any work which contains patentable inventions.

2. The employee and supervisor must consider commercial potential. The guidelines in the “conditions for limited commercial potential” may guide you. Should there be commercial potential, go through the existing innovation disclosure processes. In particular, any work which may cover patentable inventions must be reported first.
3. If all conditions are satisfied, you, in consultation with your PI, supervisor, or project leader (whichever is applicable) and any funder/client requirements, may choose to release the work. Should the supervisor or PI have a conflict of interest or possible conflict of interest, their supervisor should also be consulted.

4. Depending on funding sources, you may have more restrictions on licensing and releasing as open source. Project proposals and grant agreements may contain provisions relevant to releasing work openly. When making project proposals, consider these topics already. When in doubt, contact the relevant staff.

5. To be covered under this policy, work must be licensed under a open/open source/free software license. In case of doubt, Creative Commons, Open Source Initiative, and Free Software Foundation approved open source licenses are considered acceptable. See below for some license recommendations.

6. All warranty must be disclaimed. The easiest way of doing this is by choosing an appropriate license. Practically all of them disclaim warranty.

7. All authors must consent to the release terms.

8. The employee should not transfer an exclusive license or ownership to a third party. Aalto maintains the right to relicense and use internally, commercially, or re-license should circumstances change.

9. Employees should acknowledge their Aalto affiliation, if this possible and within the community norms.

10. This right should not be considered Aalto officially releasing any work, but allowing the creators to release it in their own name. Thus, Aalto does not assume liability or responsibility for work released in this way. Copyright owner/releaser should be listed as the actual authors.

11. Employees are responsible for ensuring that they have the right to license their work as open source, for example ensuring that all included software and data is compatible with this license and that they have permission of all authors. Also the release must be allowed by any relevant project agreements. Should you have any doubts or concern, contact Innovation Services.

To apply this to your work, first receive any necessary permissions. In writing, by email, is sufficient. Apply the license in your name, but list Aalto University as an affiliation somewhere that makes sense. Do not claim any special Aalto approval for your work.

For clarity, raw official text is separate from the guidance on this page. Current approvals: Department of Computer Science (2017-03-17).

How to run a good open-source software project

One of the largest benefits to open source is having a community of people contributing back to you. To do this, you need to have a good environment. Open development, good style and a basic contribution guide, and encouragement is the base of this. Eventually, this section may contain some more pointers to how to create this type of community. (TODO)

References

- CSC open source policy, with similar practical effects to what we have here.
- Aalto IP guide: FI EN: contains evidence that this policy is applicable to all Aalto.
- Aalto Innovation Services: https://innovation.aalto.fi/
- Choosing an open source license: https://choosealicense.com/
- Aalto copyright advice: http://copyright.aalto.fi/
- Practical guidelines for Open Source Projects: forthcoming, 2017
4.1.16 Standalone Matlab

General matlab hints: http://math.aalto.fi/opetus/Mattie/MattieO/matlab.html

Installation and license activation on staff-owned computers

Matlab academic license permits installation on home computers for university personnel. Triton MDCS workers are available to anyone with a Triton account, which means the workers can be utilized from personal laptops as well.

Download image

Log into http://download.aalto.fi/ with your Aalto account. Look for the link Software for employees’ home computers which will take you to the Matlab download links. Download the UNIX version for Linux and OSX or the separate separate image for Windows.

The ISO image can be burned on a DVD or mounted on a virtual DVD drive.

- Windows: Use MagicDisk or Virtual CloneDrive OR burn the image on DVD. Double click on setup.exe icon.
- Linux:

```
# sudo mkdir /mnt/loop
# sudo mount -o loop Download/Matlab_R2010b_UNIX.iso /mnt/loop
# sudo /mnt/loop/install.sh
```

- Mac OS X: Double click on InstallForMacOSX.app icon.

Installation steps

Select the installer options as shown in the screenshots.

Mathworks account is required to continue with the installation.

- Enter your account information in the installer to log in. If the password has been lost, Click on the Forgot your password? option to receive your password in email. OR
- Register to Mathworks with the installer.
  1. Click on I need to create an account.
  2. Enter your name and email address. To be recognized as Aalto academic user the email address must end in one of aalto.fi, tkk.fi, hut.fi, hse.fi, hkkk.fi or uiiah.fi domains.
  3. The installer will ask for an activation key, which is shown here in the last screenshot.

You may leave out unnecessary toolboxes and change the installation location. Remember however, that the Parallel Computing Toolbox is necessary to run any Matlab batch jobs on Triton.
Install Triton-MDCS integration scripts

Continue MDCS setup from Matlab Distributed Computing Server.

FAQ

Matlab freezes with Out of Memory errors

Q: Matlab freezes and I get errors like this. What to do?:

```
Exception in thread "Explorer NavigationContext request queue" java.lang.OutOfMemoryError: GC overhead limit exceeded
   at com.mathworks.matlab.api.explorer.FileLocation.<init>(FileLocation.java:89)
   at com.mathworks.matlab.api.explorer.FileLocation.getParent(FileLocation.java:126)
   ...
```

A1: Add more memory in Home -> Preferences -> General -> Java Heap memory

A2: Can you free up memory in your code sooner using the clear command? https://se.mathworks.com/help/matlab/ref/clear.html

GPU acceleration?

Q: is there functional GPU acceleration? Does the acceleration even work?

A: run code:

```
>> g = gpuDevice;
>> ng
```

A2: Just query some feature:

```
>> fprintf('%s
', g.ComputeCapability)
```

a3: Show multiple devices if found:

```
>> for ii = 1:gpuDeviceCount
   g = gpuDevice(ii);
   fprintf(1,'Device %i has ComputeCapability %s \n', ...
   g.Index,g.ComputeCapability)
end
```

4.1.17 Overleaf

Aalto provides an professional site license to all the community. For more information, see https://www.overleaf.com/edu/aalto.

In order to link yourself to Aalto, you must register for and have an OrcID [wikipedia]. An OrdID (“Open Researcher and Contributor ID”) is some permanent ID which is used for linking researchers to their work, for example, some journals require linking to an OrcID. OrdID can be accessed directly with your Aalto account.

TODO: determine exact procedure and update here
Aalto rates overleaf as for “public” data. This doesn’t mean that Overleaf makes your data public, but just that Aalto can’t promise security. In reality, you decide if Overleaf is secure enough. If there is some legal requirement for security, you probably shouldn’t use Overleaf. If there is a collaborator requirement for security, then you must make your own choice if Overleaf is suitable.

### 4.1.18 CodeRefinery

The NeIC sponsored CodeRefinery project is being hosted in Otaniemi from (previously we had one in Otaniemi from 12-14 December). We highly recommend this workshop. (note: It is full and registration is closed).

If you have an Aalto centrally-managed laptop, this page gives hints on software installation. You have to use these instructions along with the CodeRefinery instructions.

**Note:** These are only for the Aalto centrally managed laptops. They are not needed if you have your own computer you administer yourself, or if you have an Aalto standalone computer you administer yourself.

**Warning:** You should request primary user rights early, or else it won’t be ready on time and you will have trouble installing things. For Windows computers, request a wa (workstation admin) account.

### 4.1.19 Linux

You need to be primary user in order to install your own packages. Ask your IT support to make you if you aren’t already. You can check with the `groups` command (see if you are in `COMPUTERNAME-primaryuser`).

Install the required packages this way. If you are primary user, you will be asked to enter your own password:

```
pkcon install bash git git-gui gitk git-cola meld gfortran gcc g++ build-essential
˓→snakemake sphinx-doc python3-pytest python3-pep8
```

For Python, we strongly recommend using Anaconda to get the latest versions of software and to have things set up mostly automatically.

You should install Anaconda to your home directory like normal (this is the best way to get the latest versions of the Python packages). If your default shell is `zsh` (this is the Aalto default, unless you changed it yourself), then Anaconda won’t be automatically put into the path. Either: copy the relevant lines from `.bashrc` to `.zshrc` (you may have to make this file), or just start `bash` before starting the Anaconda programs.

Jupyter: use via Anaconda.

PyCharm: the “snap package” installer requires root, which most people don’t have. Instead, download the standalone community file `.tar.gz`, unpack it, and then just run it using `./pycharm.../bin/pycharm.sh`. The custom script in `/usr/loca/bin` won’t work since you aren’t root, but you can make an alias in `.bashrc` or `.zshrc: alias pycharm=... (path here)`.

Docker: you can’t easily do this on the Aalto laptops, but it is optional.
4.1.20 Mac

You also need to be primary user to install software.

If you are the primary user, in the Software Center you can install “Get temporary admin rights”. This will allow you to become an administrator for 30 minutes at a time. Then, you can install .dmg files yourself (Use this for git, meld, cmake, docker).

Anaconda: you should be able to do “Install for me only”.

Xcode can be installed via the Software Center.

Jupyter: use it via Anaconda, no need to install.

4.1.21 Windows

You should request a workstation-admin account (“wa account”), then you can install everything. Note: these instructions are not extensively tested.

Git and bash can be installed according to the instructions.

Visual diff tools: Needs wa-account.

Mingw: Not working, but seems to be because of download failing.

Cmake: Needs wa-account.

Docker: untested, likely requires wa-account.

4.1.22 Zulip

See also:

Instructors, see the relocated instructor page at Zulip for instructors.

Zulip is a open-source chat platform, which CS hosts at Aalto as a pilot. It is used as a chat platform for some courses, and allows better student and chat privacy.

The primary distinguishing feature of Zulip is topics, which allows one to make order out of a huge number of messages. By using topics, you can narrow to a certain thread of conversation (while not hiding all the older conversations from the main view).

Zulip for instructors

Introduction

Zulip is an online discussion tool with latex support. It has been used by some Aalto teachers as an external service on individual courses. For Spring/Summer 2021, Zulip is provided by Aalto CS as a pilot solution for all School of Science departments’ course needs. The pilot refers to a) a fixed-term project with clear lifecycle needs, like in courses which start and end at certain times and after which the Zulip instance can be deleted; b) a transitional period between current state and possible production use or change to other solutions; and c) a basic solution with without all the fancy features or user interface. During the pilot users are expected to provide feedback, which will effect on the decision-making for future solutions, and the development of usability.

CS-IT hosts Zulip the chat instances for you. These chat instances are hosted at <chat-name>.zulip.cs.aalto.fi. Login to the chats is available with Aalto accounts. Email registration for external users is also possible via invitations. After logging in for the first time with an Aalto account, if no matching Zulip account was found, you
are prompted to “Register” and create one. Once the Zulip account has been created, it should be linked to your Aalto credentials.

Internal or confidential matters should not be discussed on the platform.

**Get started / request Zulip**

---

**Note:** Request a chat instance at https://webropol.com/s/zuliprequest

We are taking in chat instance requests for 2021 spring and summer courses. In general, the chat instances will be removed after the academic year has ended.

---

**Note:** This service is still in beta. You might encounter some issues. If you encounter issues, report them to CS-IT or on #zulip-support at scicomp.zulip.cs.aalto.fi

You can also give/discuss feedback, complaints or suggestions on #zulip-feedback at scicomp.zulip.cs.aalto.fi

---

**Note:** You can test out Zulip at testrealm.zulip.cs.aalto.fi. Use the Aalto login. This chat is for testing only.

---

**Configuring your organization**

Below are listed the most important settings found under *Manage organization* in Zulip. There is no easy way for us to enforce these, so it is your responsibility as organization owner or admin to make sure they are set correctly. Make sure any owners/admins you appoint are aware of these as well.

**Note:** Settings that are not mentioned here, you can configure to your liking. However you should still exercise care, since you are responsible for the service and safety of your user’s data. If you would like advice, please ask us.

**Organization settings / Video chat provider**

- Set to *None*
- The default provider (Jitsi) has not been evaluated or approved by Aalto
- Integration with Aalto Zoom may come later on

**Organization permissions / Invitation settings**

Do not set both “Organizational Premissions→Invitations = not required” and “Authentication methods→Email = enabled” at the same time.

You can allow signup by Aalto account or *any* email. You can allow anyone to signup or make it invitation only. But you *can not* set “Anyone with Aalto account may signup without invitation, but by email you must be invited” (Zulip limitation). So, we have to work around this, otherwise bots and random people might join in your chat. If the chat needs to include external users, make it invite only.

The exact questions and answers:

- Are invitations required for joining in the organization?
  - If you are only allowing Aalto Login (see ‘Authentication methods’): Can be set to *No*, . . . (But still, anyone with Aalto account can join)
– If you are allowing external users/email registration (see ‘Authentication methods’ below): Set to Yes, only admins can send invitations. (You can invite people via their Aalto email address for Aalto login)

Organization permissions / Who can access user email addresses

• Set this to Admins only or Nobody

Organization permissions / Who can add bots

• Set to Admins only

• Consult CS-IT before deploying any bots

Authentication methods

• AzureAD
  – This is Aalto Login and should be enabled

• Email
  – This allows users to register using an email address
  – We cannot allow random people or bots to register freely
  – If you enable this, make the chat invitation only as described in ‘Invitation settings’ above, for the reason described there.

Users

• You can manage users here.

• Please be careful with who you assign admins/owners. These roles should be only given to course staff

Practical hints

There is a fine line between a discussion platform and chat, normal chat and topic-based chat, and chaos and order. Here, we give suggestions for you, based on what other teachers have learned.

• Topics (basically, like subject for a message thread) is the key feature of Zulip. It is explained more below, but basically keeps things organized. If you don’t want to do that or it doesn’t match your flow, you won’t like the model.

• Read the guidelines for students to see the importance of topics and the three ways to use Zulip, and how we typically manage the flood of information in practice.

• Give these guidelines to your students (copy and paste from the student page).

• Consider why you want a course chat.
  – Do you want a way to chat and ask questions/discuss in a lower-threshold platform than forum posts? Then this could be good.
  – Do you want a Q&A forum or support center? Then this may work, but would MyCourses be a better forum?
  – Do you want a place for students groups to be able to chat among small groups?
  – Do you mainly want announcements? Then maybe simply use MyCourses?

• Create your channels (“streams”) before your students join, and make the important ones default streams (this is done under “Manage organization”), so that everyone will be subscribed (the “join stream” is not obvious once you get to hundreds of people!)
– If you do create a new default stream later, use the “clone subscribers” option to clone from another default stream, so that everyone will be subscribed.

– Some common streams you might want are general, announcements, questions. Some people have one stream per homework, exam, and/or task.

• If you want a Q&A forum, make a stream called questions and direct students there.

– Remind students to make a new topic for each new question. This enables good follow-up via “Recent topics”

– If students don’t make a new topic (or a topic goes off-track), edit the message and change the topic (change topic for “this message and all later messages”). Then, you keep questions organized, findable, and trackable.

– You can use the “forum bot” (https://github.com/AaltoSciComp/zulip-forum-bot). This is still a work in progress we have, but the basic idea is that you react to a message with check mark (✓), and then the topic gets renamed to include “✓” at the beginning, so you can clearly identify answered and unanswered questions in the “Recent topics” view. We will add more features as people request. Please ask our help when deploying bots.

– If you don’t want to be answering questions in private message (who does? . . . it leads to duplicate work), make a clear policy on either reposting the questions publicly yourself (without identification), or directing the students to repost in the public stream themselves.

• If you want to limit students to not be able to do anything, you can consider disabling:

  – Adding streams, adding others to streams (if you want people to only ask and not make their own groups).
  – Disable private messages (if you really don’t want personal requests for help).
  – Adding bots, adding custom emojis.
  – Seeing email addresses. Changing their name.

• On the other hand, you might want to “allow message editing” to a much longer period and allow message deleting. For Q&A these are quite useful to have.

• You can use the /poll [TITLE] command to make lightweight non-anonymous polls. For anonymous polls, someone has used a bot called Errbot, but we don’t currently know much about that.

Things we know are missing (see also the student page):

• More fine-grained permissions for TAs.

• Support for bots and other advanced features (more like permission to recommend them).

Basics

Streams and Topics

In Zulip, discussions are organized in streams, which are further divided into topics.
Views

Main views

The left **sidebar** let’s you narrow down messages that are displayed, you can select:

- **All messages**, to see everything that is being posted efficiently.
- **Recent topics**, to see which topics have new information.
- **Different streams and topics**, to narrow down to a specific stream or topic.

**Recent topics** is good to manage a flood of information (see what’s new, click on relevant stuff, ignore all the rest).

**All messages** is better when you are caught up and want to make sure you don’t miss anything. Viewing **single topics**
and streams is good for catching up on something you don’t remember.

Of course, everyone has their own ways and workflows so you should experiment what works best and which views are useful for you.

**Message Pane**

In the middle of your screen, you have the **Message Pane**, where the messages are shown.

![Message Pane](Image)

Fig. 6: **Message Pane.** This is the basic view of messages. You can click on various places to narrow your view to one conversation or reply.

**Selecting visible topics**

Not all streams are visible in the sidebar by default.

Click the gear icon above the channel list in order to see all available streams and select which ones you want to participate in. It is good to occasionally look at this menu in case new streams are added.
Hints on using Zulip efficiently

How to ask a question

Seems obvious, doesn’t it? You can get the best and fastest answers by helping to keep things organized. These recommendations are mainly for Q&A-forum type chats.

- First, search history to see if it has already been asked.
  - If so, click on the topic name. You will narrow your view to see that entire conversation.

- If your question isn’t answered yet, but is a follow up to an existing topic, click on a message in that topic. Then, when you ask, it will go to that same topic as a follow-up, and anyone else can narrow to see the whole history.
  - Unlike other chats, your message will not get lost, and people will both see that it is new and can see the history of that thread.
  - Your course can say what the threshold for “new topic” is. Maybe they would have one topic per question pre-created or something clever like that.

- If you don’t find anything relevant to follow up on, make a new topic.

Fig. 8: Replying to an existing topic.

- Select the stream you want to post to (whatever fits best).
– Click “New topic”.
– Enter the topic name down below: a few words, like an email subject. For example, week 1 question 3, integrals of complex functions, exam preparation.
– Enter your message and send.

Others (or you...) can split or join topics if they want by going to “edit message”, so there is no risk of doing something wrong. Don’t worry, just ask!

By being organized, you can get both the benefits of quick chat with the organization of not missing anything.

Other hints

• You can format your messages using Zulip markdown.
• “Mute a stream” (or topic) is useful when you want to stay subscribed but not be notified of messages by default. You can still find it if you click through the sidebar.
• The desktop and mobile apps can support multiple organizations. At least on mobile apps, switching is kind of annoying.

Apps

There are reasonable applications for most desktop and mobile operating systems. These don’t send your data to any other services.

The mobile applications work, but may not be the best for following a large number of courses simultaneously. We can’t currently make improvements in them.

Open issues

We are aware of the following open issues:

• It is annoying to have one chat instance per course (but it seems to be standard in chats these days).
• There are no mobile Push notifications (since Aalto Security won’t let us turn them on).
• Likewise with built-in video calls (via https://meet.jit.si or Zoom).
• Various user interface things. But Zulip is open-source, so feel free to contribute to the project...

Cheatsheets: CS, Data.

Fig. 9: Making a new topic.
In this section, you can find some information and instructions on data management. Concrete information: main Aalto services and global services. Main Theoretical information: Aalto-specific summary and Aalto’s Research Data Management pages.

5.1 Data

Data binds all of today’s research together. Even if you don’t consider yourself to do data-based research, the results of your work becomes data before it is published. The highest levels of funding agencies are beginning to demand good data management and openness. Knowing how to manage data is probably one of the most important untaught modern skills.

It’s not just “get it done”: there are good and bad methods of managing data. For example,

- A **bad strategy** is to store everything in one folder on your own laptop: there’s a very high chance that you will someday lose it all. A **better strategy** is to use a secure centralized service - preferably an Aalto service, since you get guaranteed support for free.

- A **bad strategy** is for everyone to do their own thing and put no effort into recording what they have done: in five years, when the group is almost completely changed, that data will be unusable. A **better strategy** is to make sure things are documented and archived as soon you get them (and keep this up to date), so that the data can continue to serve you in the future.

- A **bad strategy** is to assume all data is proprietary Aalto information: eventually funders will demand more and you won’t be prepared, and Aalto will remain an island, instead of a hub that others want to work with and build on. A **better strategy** is to always consider openness, licensing, and privacy from the start (even if you don’t do it right away), and always separate data based on level of confidentiality so that you can share or open later.

You can find more formal information at the Aalto Research Data Management pages, and here we focus on the practical side of things.

5.1.1 Applying for funding

When applying for funding, you may need to submit a data management plan (DMP) along with the grant application. For hints on making one, see our data management plan page or the Aalto Research Data Management pages. However, be aware that a grant application data management plan (“Funder DMP”) usually focuses on sounding like a grant, not being a usable work plan (“Practical DMP”). Before you start accumulating data, browse the other links on this site and make sure that you organize things well! Aalto info will only help you make a funder DMP, not organize your data during the project.

Grantwriters and the Open Science and ACRIS teams can help you with producing data management plans for funding. Science-IT can help you with funding or practical data management plans.
Data management plans

Data management plans are a catchphrase these days, mainly because funders are requiring them now. This is for a good reason - researchers often focus on their papers, and making good use of the data gets forgotten. Funders pay a lot for research, and they want all the possible value for society.

However, it is worth doing a bit of planning about data, even aside from the required bureaucratic exercise. It is true that researchers focus on the next paper. Data has long-term value even inside Aalto, and if you don’t try hard it will get lost.

Actual plan

In this section, we outline recommend ways to use Aalto resources for different use cases.

No matter what your project, you want to start by thinking how you will handle your data (this can be “real data”, notes, code, papers, etc). This will make sure that your team works together well and doesn’t end up with a big mess in a few months - or that you can’t work together because you can’t share information. For this, see the A4 DMP template. This site is focused on practical DMPs.

- Suggested DMP for large experimental data (TODO)
- Suggested DMP for simulations or computer-generated data (TODO)
- Suggested DMP for data from humans (surveys, interviews, etc) (TODO)

Funder plan

There are plenty of other good resources about making funder DMPs.

- At Aalto, the RIS grantwriters have taken responsibility for helping to make good funder DMPs.
- The Aalto RDM pages have a subsection dedicated to data management plans.
- The DMPTuuli is a combination template, instructions, and web form which makes it easy to do the mechanical assembly of DMPs. They also have public docx/pdf templates which can be used even without the web form. Aalto recommends this service, though be aware it helps you fill out a form, not plan your work.

As some concrete suggestions:

- Funders are especially concerned about sharing, preservation, reproducibility, and dissemination but probably can’t evaluate too much about the practical side of things.
- You can mention that you will follow the Aalto RDM policy, which covers mainly opening and licensing. The policy still allows you to make your own choices, but it sounds quite good if you refer to it and say you will follow it.
- For data storage considerations, you can say that your department/Science-IT provides data storage services (for Science-IT departments) and has a data storage policy which you will follow: citation and/or full text.
Help! I need a DMP right now!

If you are reading this, you probably have a grant deadline and you need to do something right now. Use the resources above, but here is some more advice:

- Read the data management outline on this site. You should be able to pull many of the practical pieces (storage, confidentiality, archiving, etc) from here. Read this first!
- Read the Aalto-level guidelines. These are quite abstract and high level, and might tell you what people think is important but not tell you how to do stuff.
- To internally organize things, you could start with the A4 DMP template. This can’t be used for something you submit, but lets you know the big picture. If you fill this out first and give it to someone, they can guide you in making the next version.
- Use the DMP Tuuli tool to prepare the DMP. It just makes a final document you can download (you could do the same using a word processor), but breaks everything down into a nice form.
  - If you don’t like the idea of a web form, the templates seem to be available publically, too. These seem to have roughly the info as the DMPTuuli web forms.

Why do they want DMPs? What should it include? Answering these will help you to know what to write, since there is not near enough room to make a plan that contains everything you need to know personally:

- The main purpose is to make sure that other researchers can use your data as easily as they can use your published papers. Can other researchers access your data? Can your results be reproduced?
- Most likely, whoever is reading doesn’t care that much about the actual day to day data storage and so on, but more of the big picture: licensing, opening, archiving, sharing, preserving, expanding, securing.
- If you produce your own data, how can others use it? Funders want open, but by giving good justification you can do whatever you need. If the data comes from others, then can you re-distribute (even for validation) or would others need to request it from the source?
- How software you make related to data processing (and really all software) will be handled. Even if data can’t be released, software can be open sourced which allows reproduction of results and some sort of validation.
- How you preserve data for future use: both for you, and for others. This is especially important. Also, how will data be understandable in 50 years? Is the program that will read it gone? Do you have a README? Is your data in a field-specific standard structured format? Is it opened and does it go into an archive which will be around in 50-100 years (anything managed by you or Aalto specific isn’t a credible option for this)?
- You should mention how you will follow the “Aalto Research Data Management Policy and related guidance”. The policy just says “you will make strategic decisions”, so sounds good to the funder while not binding you to anything.
- For storage, organization, confidentiality, etc, you can say you will follow the Science-IT data management policy. This isn’t requirements for you, but the default services we offer for data storage (designed to keep data safe and secure, and uuushareable). It also sounds good to say. (see the outline)
Model Academy of Finland DMP

You can see the Academy’s detailed info in their supplement. This guide isn’t to replace their guidelines (there is a lot there that isn’t duplicated here), but make it clear what the Aalto correspondences are. You can also see the Aalto guidelines, but this is also a bit abstract to be immediately usable.

With all the time spent on writing your plan, don’t forget to do something useful, too.

1. General description of the data
   - No specific extra advice here - see academy guidelines.

2. Ethical and legal compliance
   - For identifiable human data, say that you will follow the Aalto personal data policy. In particular, data will only be stored only on systems meeting the Aalto guidelines for personal data storage. Preferable, store this on the department network drives only - not on personal computers. You can request ethical evaluation from the Aalto Research Ethics Committee. In Finland, this is required in quite few cases, but publishers are requiring this more and more often. Thus, you may want to check your journal requirements and request ethical evaluation anyway.
   - Data always will be made available under the Aalto data management policy. (You can commit to this, because the policy only says you should make decisions “strategically” so there are actually no obligations.)
   - Software will be made open source if it matches the criteria under the Aalto open source policy. If software exceeds that criteria, there will be discussions with Aalto innovation services for commercialization or licensing.
   - There are plenty of other intellectual property concerns which I can’t go into here, and you need to study yourself. Aalto Research and Innovation Services has lawyers which can help with this - you can consult in advance or say you will use them.

3. Documentation and metadata
   - It is harder to comment on this because it is so field-specific. Make sure you have READMEs and documents.
   - Everyone talks about “metadata” but this is such a broad term that it is essentially meaningless. I personally put this into three types:
     - Cataloging: You can say that the metadata required by your repository will be used.
     - Necessary to understand: you will use README files, use formats that are self-describing such as CSV files with useful headers and comments, include code, and whatever is needed to make someone understand the data later (including yourself).
     - Necessary to automatically process: data should be automatically usable with the least amount of manual effort. This is highly domain-specific, and depends on if your domain already has standards to make this possible. Use the best possible practices here, taking into account cost vs benefit.

4. Storage and backup during the research project.
   - Aalto really excels here. Basically, just use the Aalto network drives. This storage is large, free, shareable, snapshotted, backed up to an ofsite datacenter. Access is controlled via Aalto accounts plus unix groups. If people need to make other copies (and it’s allowed for security reasons), they can. Big data is stored on Triton cluster from which it has direct access to any computational power you may need.

5. Opening, publishing, and archiving the data after the research project.
   - This gets more abstract, and really depends on what you want. There are many options, and maybe it is best to consult the Aalto page on this, though it’s again rather abstract.

Chapter 5. Data management
• You can check the services page to see what common services are available. If you don’t have any more specialized repository to use, Zenodo is a good choice. Always prefer a specialized, domain-specific repository if you can. Don’t say it is archived on Aalto resources, since you or Aalto can’t commit to hosting things or the long term.

• You can say that organization of data is a part of research, though the extra requirements needed to open are small. Give some estimate of the total/extra amount of work needed.

5.1.2 During the project

Make sure that you manage data well - just think, your data is possibly worth more than all your other devices combined. Check out the core lessons corelessons (non here yet) to learn of the most common problems, and see if any of them apply to you.

You may want to read our welcome to researchers and outline of data management at Aalto pages. For specific Aalto storage services, see Data storage, and for other options see the general services page.

We recommend that each project or group gets a network drive, which is used as the centralized place for data storage, safekeeping, and possibly daily work. See the outline of data management at Aalto page.

Data storage services available

This page provides a list of common data storage services, and can help you select the right service for the type of data you have (see Data organization). But before we talk about services, you have to consider what your needs are.

Types of data

There are different broad categories of data:

• Code/papers drafts/: These are absolutely critical, but quite small. You want a full history that is easy to use, too. Put in version control and in Aalto gitlab.

• Original data: Your original, irreplaceable data. You want this in two places: a fast, large, available place for day-to-day work, and also somewhere backed up for a fairly long time.

• Intermediate working files: This is what you get when you run code on original data. It’s OK if this is lost, because you have the code and original data to re-create it, right? It can go in the large, fast location.

• Final published results/data: You want this backed up and available for a very long time (forever?). Put in an open-access repository such as Zenodo. Once it’s in the archival, backups should be done there.

O = good, x = bad

<table>
<thead>
<tr>
<th></th>
<th>Large</th>
<th>Fast</th>
<th>Confidential</th>
<th>Frequent backups</th>
<th>Long-term archival</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td></td>
<td></td>
<td></td>
<td>OO</td>
<td>OO</td>
</tr>
<tr>
<td>Original data</td>
<td>O</td>
<td>O</td>
<td>O?</td>
<td>OO</td>
<td>OO</td>
</tr>
<tr>
<td>Intermediate files</td>
<td>OO</td>
<td>OO</td>
<td>O?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final results/open data</td>
<td></td>
<td></td>
<td></td>
<td>OO</td>
<td></td>
</tr>
</tbody>
</table>
There are different qualities we want in filesystems: large, fast, confidential, highly available, backed up, mounted everywhere, lasts forever. It is expensive to have all of these together, so there are different places with different benefits. It is up to you to balance their use so that you can accomplish what you need. Compare this table to the types of data above. Use the right place for the right data.

You often need to use different types of services, for example version.aalto.fi for day to day code management, but archive to Zenodo at the end of a project.

\( O = \) good, \( x = \) bad

<table>
<thead>
<tr>
<th>O</th>
<th>Large</th>
<th>Fast</th>
<th>Confidential</th>
<th>Back-ups</th>
<th>Long-term archival</th>
<th>Shareable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triton scratch</td>
<td>OO</td>
<td>OO</td>
<td>O</td>
<td>x</td>
<td>x</td>
<td>O</td>
</tr>
<tr>
<td>Triton work</td>
<td>OO</td>
<td>OO</td>
<td>O</td>
<td>x</td>
<td>x</td>
<td>O</td>
</tr>
<tr>
<td>Triton home</td>
<td>x</td>
<td>O</td>
<td>OO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local disks</td>
<td>O</td>
<td>OO</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ramfs</td>
<td>OO</td>
<td>OO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Depts /m/.../project</td>
<td>O</td>
<td>O</td>
<td>OO</td>
<td>OO</td>
<td></td>
<td>O</td>
</tr>
<tr>
<td>/m/.../archive</td>
<td>O</td>
<td>O</td>
<td>OO</td>
<td>OO</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Aalto Aalto home</td>
<td>OO</td>
<td>OO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aalto work</td>
<td>O</td>
<td>O</td>
<td>OO</td>
<td>OO</td>
<td></td>
<td>O</td>
</tr>
<tr>
<td>Aalto teamwork</td>
<td>O</td>
<td>O</td>
<td>OO</td>
<td>OO</td>
<td></td>
<td>O</td>
</tr>
<tr>
<td>Aalto laptops</td>
<td>x</td>
<td>x</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aalto webspace</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>version.aalto.fi</td>
<td>OO</td>
<td>OO</td>
<td>O</td>
<td></td>
<td>OO</td>
<td></td>
</tr>
<tr>
<td>ACRIS</td>
<td>O</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eduuni</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aalto Wiki</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finland Funet filesender</td>
<td></td>
<td>O</td>
<td></td>
<td></td>
<td>OO</td>
<td></td>
</tr>
<tr>
<td>CSC cPouta</td>
<td>O</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td>O</td>
</tr>
<tr>
<td>CSC Ida</td>
<td>OOO</td>
<td>x</td>
<td>OO</td>
<td>O</td>
<td></td>
<td>O</td>
</tr>
<tr>
<td>FSD</td>
<td>OO</td>
<td>O</td>
<td>OO</td>
<td>O</td>
<td></td>
<td>O</td>
</tr>
<tr>
<td>Public github</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>OO</td>
</tr>
<tr>
<td>Zenodo</td>
<td></td>
<td>OO</td>
<td></td>
<td>OO</td>
<td></td>
<td>O</td>
</tr>
<tr>
<td>Google drive</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OneDrive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Own computers</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Emails</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EUDAT B2SHARE</td>
<td></td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td></td>
</tr>
</tbody>
</table>
Service details

Note: This list is still under development (2018-03-07)

In general, if you need to

- **archive and open**, consider hosting data on Zenodo (and put a record of it in ACRIS, so you can get internal Aalto credit. If you have a discipline-specific repository, use that instead (with metadata still in ACRIS)
- For **day to day work** within Aalto, Aalto network drives are a good service and (different options below).
- For making a **data management plan**, DMPTuuli along with our info is good.

Science-IT services

- The filesystems by *Triton*. Primarily scratch and work, which are very large, very fast on Triton, but only for scratch data because they are not backed up.

Departments

- **CS,NBE,PHYS** provide storage logically divided into project and archive. These are the counterparts of Triton and are backed up. They are actually Aalto “teamwork”, but the departments do the day-to-day interfacing. See *Data storage*.
- See the **work** and **teamwork** notes below in the next section. In some cases, these are managed by departments.

Aalto

- See *Data storage*.
- Also information is available from Aalto ITS, some here.
- **Aalto home directories** are small and intended mainly for personal stuff. Once you leave, this data dies, so don’t put important stuff here.
- Aalto has work and teamwork storage systems. These are actually provided at the Aalto level, but how you request space, how you use them, and what the are called varies and is not always very well defined. A little bit of info at *Data storage*.
- **Aalto laptops** are not a good place to store data because they are usually not backed up, and data is not shareable. (Even if data is backed up, once you leave, no one will even be able to get access). Most people who use laptops have the most valuable data stored on network drives.
- **Aalto webspace** can share data. See *Data storage*. This isn’t suitable for archival or long-term anything, since it is tied to user accounts. If you want to share here, maybe you could do a bit more work and handle it forever at Zenodo?
- [https://version.aalto.fi](https://version.aalto.fi) is the Aalto Gitlab. It is used for small version controlled files. It is a great place for day to day work of private files, but not for permanent archival. See *Aalto Gitlab*.
- **ACRIS** is the Aalto “research information system”, meaning it’s a record of things that everyone is doing research-wise. You should make records for datasets there as a research output. (*ACRIS + research data instructions*)

Summary: try to host the actual data elsewhere, but always make a report of the data in ACRIS so you get credit.
ACRIS has support for storing data itself, but that isn’t recommended most of the time since ACRIS in its current form isn’t guaranteed to stay around forever. However, if data needs to be kept internal, it might be OK since you can set confidentiality and share with certain people. However, you should always make a report of your datasets in ACRIS even if they are hosted elsewhere, so that you can get academic credit for it.

What data sets should be included in ACRIS? We think: a) anything that is independently published with DOI. b) any paper which serves as a formal dataset description in a data journal, even if there is also an entry as an ACRIS article. c) any paper which serves as an informal dataset description.

As for different roles: creator=who is involved in creating it, distributor=who can be contacted about access (if not public), owner=who has ultimate responsibility (often the PI but project dependent).

• Eduuni is a Finnish service for educational collaboration. It’s reported to be more secure than either Google Drive or OneDrive, but we know of few people who use it.

• The Aalto Wiki is sometimes mentioned as a place to store data. It’s really better for collaboration, but you can put little bits of data there if you want.

Finnish services

• The FUNET filesender (https://filesender.funet.fi) can share files with others. You log in with your Aalto account, and then you can upload files and send a link by email. Or, you can send an email that allows others to upload. Run by CSC and recommended for sharing (instead of email).

• IDA, Etsin, and AVAA are CSC-provided services (funded by the ministry as part of the Open Science project, ATT), which provide some data services to researchers.
  – Etsin is the Finnish metadata catalog. The intention is that all research data eventually gets cataloged here (open or not), but we are quite far from that goal. Ideally, there would be bidirectional imports to and from ACRIS (the Aalto system) and other repositories, but it’s not there yet. We should recommend that you make a note of your data here, but realistically do ACRIS and wait for a link.

  – IDA is a storage service. (instructions) It is based on iRODS, a data management layer on top of filesystems. Thus, you have to access it using a special API, command line interfaces, or other tools. Because of this, the learning curve is very steep. Currently, we think IDA would be good if your university doesn’t provide large enough free, properly backed up storage that is shareable within Finland. For long-term public storage, Zenodo is probably overall easier to use. We have some practical notes on using it here, because it takes quite a few steps to get started.

It is said to be a safe place to store your data, but if you read closely a different “long-term preservation” service is coming, so IDA isn’t that. IDA might have a use case for confidential data which can’t leave Finland, but it says it claims it is not suitable for such. They also say that metadata “shall” be added, which makes you think it is only for data which is prepared enough for putting in Etsin.

If you are dealing with a large amount of data and want to use an API to handle it, this could be good.

IDA is being renewed in 2018, and will need reevaluation then.

  – AVAA is basically a merging of IDA and Etsin. You can set some metadata in IDA so that your data is available via the web. There are some instruction in the IDA user guide (browser, command line). Overall, having to use three different services for publishing a file takes a fair amount of work, so if you want to open data, Zenodo is faster.

• The FSD Finnish Social Data Archive / Tietoarkisto is run from the University of Tampere. It is a full-service archive for social data, so they can help in data preparation and curation. It is one of the few places in Finland allowed to archive personally identifiable data.

• DMPTuuli (dmptuuli.fi) is a service for making data management plans. It is primarily targeted at funder DMPs, so it won’t help you plan your actual research (and even for funder DMPs, you need to know what to say). You
can check our *data management plans* page, including the “emergency DMP” section. Aalto also has a little bit of guidance.

**EU services**

- **Zenodo** ([https://zenodo.org](https://zenodo.org)) is a long-term data repository. It is the largest (thus the most stable long-term) and also has a great user interface. You get a DOI if you archive here. We recommend this service unless you have another domain-specific repository that fits your data better. If you publish data here, also make a metadata entry in ACRIS (see above).

  Zenodo is a good service, but there is little curation, so you need to make sure that your data is described well (both in the structured catalog information and within the data, so that it is usable).

  When you put data in Zenodo, also make an ACRIS dataset entry linked with the DOI.

- **EUDAT** ([https://eudat.eu](https://eudat.eu)) provides a lot of different services: B2share is a lot like Zenodo, but smaller and last we checked the user interface wasn’t as good (and it didn’t provide DOIs). B2Drop is a Dropbox-like file sharing service (powered by nextcloud), which can be quite nice. B2Find is a metadata catalog that lets you search for data. The other services are mostly target to other large infrastructures. *(EUDAT will be re-evaluated in 2018)*

**Global services (with special Aalto support)**

- **Google Drive** is a cloud storage solution (but you probably already knew that...). You can register your Aalto account as a Google account, which gives you unlimited storage (note that this does not mean your personal account gets unlimited... a Gsuite account does. This account ends when you leave Aalto, so this should not be used for permanent storage). You have to enable your account using ITS instructions here. Access the Aalto Google Drive from [https://gdrive.aalto.fi](https://gdrive.aalto.fi). This service can be great for sharing, but because it is tied to your Aalto account, you should not store valuable research data here.

  Google Drive (organizations only) has a “shared drive” concept, which will allow you to put data into groups which can easily be inherited as time goes on, even if the original people move on.

- **Microsoft OneDrive** is like Google Drive, and Aalto has a special agreement. You can find instructions from ITS here. Theoretically, OneDrive has a higher security rating than Google Drive, but it is still not suitable for legally confidential data.

- **Dropbox** is like the above two. You can find ITS instructions here. You can sign up using a detailed procedure there. Again, this isn’t suitable for confidential/personal data, and everything vanishes once you leave Aalto.

**Global services**

- **Github** is a code-sharing and collaboration service (using git, obviously). If you have an open source project, this is a well-known place to put it. The only downside is if you have objections to proprietary services. Github should not be used as a permanent archive, but there is Zenodo integration so that your code can be archived permanently (and even has integration with the Github “release” feature).

  This is by no means a complete list...
Data organization

How should data be stored? On the simplest level, this asks “on what physical disks”, but this page is concerned about something more high-level: how you organize data on those disks.

Data organization is very important, because if you don’t do it early, you end up with a epic mess which you will never have time to clean up. If you organize data well, then everything after becomes much easier: you can archive what you need. Others can find what they need. You can open what you need easily.

Everything here applies equally if you are working alone or if you are part of a team.

Organize your projects into directories

Names

As simple as it seems, choosing a good name for each distinct workspace is an important first step. This serves as an identifier to you and others, and by having a name you are able to refer to, find, and organize your data now and in the future.

A name should be unique among all of your work over all your career, and also unique among all of your colleagues, too (and any major public projects, too). Don’t reuse the same names for related things. For example, let’s say I have a project called xproject. If I track the code separately from the data, I’d have a different directory called xproject-data and the main projects refers to the data directory, instead of coping the data.

How many named workspaces should you have for each project? It depends on how large they are and how diverse the types of data are. If the data is small and not very demanding, it doesn’t matter much. If you have large data vs small other files, it may be good to separate out the data. If you have some data/code/files which will be reused in different projects, it makes sense to split them. If you have confidential data that can’t be shared, it’s good to separate them from the rest of the data.

Names should be usable and directory names and identifiers. Try to stick to letters, numbers, -, and _ - no spaces, punctuation, or symbols. Then, the name is usable on repositories and other services, too.

Good names include MobilityAnalysis, transit, transit-hsl, and lgm-paper. Bad names are too general given their purpose or what else you might do.

Each directory’s contents moves together as a unit, as much as possible.

Organizing these directories

You should have a flat organization in as few places as possible. For example, on your laptop you may have ~/project for things for the stuff you mainly work on and ~/git for other minor version controlled things. On your workstations or servers, you may also have /scratch/work/username which is your personal stuff that is not backed up, /m/cs/project/hostname/username/ which is backed up, /local which is temporary stuff on your own computer, and so on. The server-based locations can be easily shared among multiple people.

Your structure should be as flat as possible, without many layers in each directory. Thus, to find a given project, you only need to look inside each of the main locations above, not inside every other project. This allows you to get the gist of your data for future archival or clean-up. When two directories need to refer to each other, you have them directly refer to each other where they are, for example use ../xproject-data from inside the xproject directory. (You can have subdirectories inside the projects).

Different types of projects go in different places. For example, xproject can be on the backed up location because it’s your daily work, while xproject-data is on some non-backed up place because you can always recover the data.
Synchronizing

If you work on different systems, each directory of the same name should have roughly the same contents - as if you could synchronize it with version control.

For small stuff, you might synchronize with version control. You may use some other program, like Dropbox or the like. Or in the case of data which has a master copy somewhere else, you just download what you need.

Organize files within directories

Traditional organization

This is the traditional organization within a single person’s project. The key concept is separation of code, original data, scratch data, and final outputs. Each is handled properly.

- PROJECT/code/ - backed up and tracked in a version control system.
- PROJECT/original/ - original and irreplaceable data. Backed up at the same time it is placed here.
- PROJECT/scratch/ - bulk data, can be regenerated from code+original
- PROJECT/doc/ - final outputs, which should be kept for a very long term.
- PROJECT/doc/paper1/ - different papers/reports, if not stored in a different project directory.
- PROJECT/doc/paper2/
- PROJECT/doc/opendata/

When the project is over, code/ and doc/ can be backed up permanently (original/ is already backed up) and the scratch directory can be kept for a reasonable time before it is removed (or put into cold storage).

The most important thing is that code is kept separate from the data. This means no copying files over and over to minor variations. Could should be adjustable for different purposes (and you can always get the old versions from version control). Code is run from the code directory, no need to copy to each folder individually.

Multi-user

The system above can be trivially adapted to suit a project with multiple users:

- PROJECT/USER1/.... - each user directory has their own code/, scratch/, and doc/ directories. Code is synced via the version control system. People use the original data straight from the shared folder in the project.
- PROJECT/USER2/....
- PROJECT/original/ - this is the original data.
- PROJECT/scratch/ - shared intermediate files, if they are stable enough to be shared.

For convenience, each user can create a symbolic link to the original/ data directory from their own directory.
Master project

In this, you have one long-term master directory for a whole research group, and members project that has many different users and research themes with in. As time goes on, once users leave, their directories can be cleaned up and removed. The same can happen for the themes.

- PROJECT/USER1/SUBPROJECT1/...
- PROJECT/USER1/SUBPROJECT2/...
- PROJECT/USER2/SUBPROJECT1/...
- PROJECT/original/
- PROJECT/THEME/USER1/...
- PROJECT/THEME/USER2/...
- PROJECT/archive/

Common variants

- Simulations with different parameters: all parameters are stored in the code directory, within version control. The code knows what parameters to use when making a new run. This makes it easy to see the entire history of your simulations.
- Downloading data: this can be put into either original or scratch, depending on how much you trust the original source to stay available.
- Multiple sub-projects: this can be
- Multiple types of code: separate long-term code from scratch research code. You can separate parameters from code. And so on...

Projects

In Aalto, data is organized into project groups. Each project has members who can access the data, and different shared storage spaces (project, archive, scratch (see below)). You can apply for these whenever you need.

What should a project contain? How much should go into the same project?

- **One project that lasts forever per research group:** This is traditional. A professor will get a project allocated, and then people put data in here. There may be subdirectories for each researcher or topic, and some shared folders for common data. The problem here is that the size will grow without bound. Who will ever clean up all the old stuff? These have a way of growing forever so that the data becomes no longer manageable, but they are convenient because it keeps the organization flat.
  - If data size is small and growing slower than storage, this works for long-term.
  - It can also work if particular temporary files are managed well and eventually removed.
- **One project for each distinct theme:** A research group may become interested in some topic (for example, a distinct funded project), and they get storage space just for this. The project goes on and is eventually closed.
  - You can be more fine-grained in access, if data is confidential
  - You can ensure that the data stays together
  - You can ensure that data end-of-life happens properly. This is especially useful for showing you are managing data properly as part of grant applications.
You can have a master group as a member of the specific project. This allows a flat organization, where all of your members can access all data in different projects.

### 5.1.3 Internal reporting

Data is a top-level research output, even if not everyone considers it valuable now. Open or not, the university wants to know what data exists. Currently, this is done via ACRIS (primary instructions). In particular, you should create a “dataset” object for data you create (it doesn’t have to be open). For some hints, for now see the ACRIS point on the services page or the ACRIS instructions on data.

### 5.1.4 Sharing and collaboration

Obviously, you will often need to share data within projects. Emailing things back and forth is rarely a good way to do things. Check other data sharing services from our services page or Aalto’s IT services for research page.

We recommend, instead of seeing this as a sharing problem, see this as a storage problem: find a place to store data which everyone can access, and share via that. This promotes long-term organization.

### 5.1.5 Archival after the project

After a project is done, you may need to store data long-term for follow-up use. You shouldn’t do this just by assuming everyone keeps their copy: people leave, and eventually that a data will get lost. The easiest and recommended way of doing this is by opening data and publishing it on a reputable worldwide archive once it is time. For the most part, the university wants to avoid creating its own internal permanent archives, because they will end up requiring large effort to maintain. It’s better to use the publically-funded and managed worldwide services.

### 5.1.6 Publication

See our list of storage services for recommendations on archival. If you don’t know what to pick (there isn’t something specialized for your field), use Zenodo and report it in ACRIS (see “internal reporting” above).

### 5.1.7 Licensing and intellectual property

Just because data is “out there” doesn’t mean it’s usable by others: big companies have ensured that data is by default closed. Luckily, it is easy to make data reusable: just add a license. There are plenty of options that can balance between “public domain, do anything” and “if you help us too”.

See our Open Source page for more info.

### 5.1.8 Other info

IDA data storage service

---

**Note:** This page is under development

**Note:** IDA has changed in 2018/2019, and these instructions may no longer be accurate.
IDA is a storage service provided by the Ministry of Culture of Education / the Finnish Open Science and Research initiative / CSC. It can be used for storing very large files securely and for a reasonably long time. Quota can be in many TB, and quota is allocated by application.

The upstream instructions can be found at https://www.fairdata.fi/en/ida/user-guide/. The upstream description can be found at https://www.fairdata.fi/en/ida/

What it is for

Main article: https://www.fairdata.fi/en/ida/

IDA is for stable research data which needs safe, somewhat long term storage. (However, it isn’t for very long term archival, another system is coming for that). It isn’t for active, day-to-day use. It can link data to permanent identifiers, store metadata, and also publish data via AVAA and make it searchable via Etsin.

If you just need large storage, Triton’s scratch is good for that. However, if you have many TBs of data, then finding a backup place is difficult (scratch is not backed up). IDA can serve that need.

IDA can also serve to make small or large data open (searchable and downloadable), via Etsin and AVAA. These three go together: IDA is storage, Etsin is search, AVAA is download server.

You automatically get a quota from Academy of Finland projects (and it says they encourage its use).

Registering and applying for space

Main article: https://www.fairdata.fi/en/ida/becoming-an-ida-user/

Everyone can apply for IDA space via your CSC account (but all IDA space is allocated to projects, not individual users). Anyone at a Finnish university can get a CSC account automatically. IDA space is

First, you need a CSC account. You can get this online via the Aalto authentication: https://my.csc.fi.

Once you have the CSC account, you need a CSC project. Only senior level staff (postdoc or above) can do this - you probably want it to be someone who will be here long term, since that is the point!. Apply for the project through the scientists user interface (SUI) (https://my.csc.fi). The SUI can be rather confusing. First, go to eService → Resources and Applications → select “Academic CSC project”. The bottom of the page then changes to an application form. Fill this out: say you need a project for IDA (or whatever). You need to wait for an email for the CSC project to be approved.

After this project is approved, you can apply for IDA storage space to be connected to this project. Go to the SUI → eService → Resources and Applications, then go to Resources → Storage → IDA Storage Service. The application form below changes to the IDA application. Select the project which will receive the resources, then fill out the application.

You will get another email with your IDA password and path once it has been approved by Aalto’s IDA contact person. This is different from the project approval email from CSC (the right email has explicit IDA usernames and passwords in it). If you do not get the IDA info email within a day or two, ping Juha Juvonen at Aalto and ask if the project has been approved.
Confidential data

In many places, CSC states that IDA is not suitable for confidential data. This is because the command line interface does not encrypt files (though I had heard that it just doesn’t by default, but maybe it could be made to). Still, since they do not intend to support confidential data, we should not count on this for the future. **However, confidential data is OK if it is strongly encrypted.**

See our page on *encryption for scientists.*

Access


iRODS (and thus IDA) is an API-based file storage service. Thus, you use separate commands to get and put files. This comes out of the fact that this is designed for very big files and flexible, long-term storage. This is not too hard - it is like using FTP or sftp. There are also mountable filesystems, however this should not be used for daily work since they are not very efficient.

Not all tools are suitable for very large files - there are some reported problems that need to be worked around. See the CSC instructions for details and hints for large files.

Note that the IDA password is different than your CSC or Aalto passwords. Don’t use your CSC or Aalto password with IDA accidentally, some of the programs (command line tool in particular) don’t seem to handle it very securely (it is stored weakly obfuscated in a file in `~/.irods`)

Browser

Through the CSC SUI, you can brows and upload files. See [https://www.fairdata.fi/en/ida/user-guide/#files-view.](https://www.fairdata.fi/en/ida/user-guide/#files-view.) This is probably not good for extremely large files.

Command line


**irods commands:** Aalto workstations and Triton have the irods command line tools (the “icommands”). Use the module system: module load irods.

**Configuration file:** You need to set up the config file (see the fairdata.fi instructions). You need a extra path in it here:

On Aalto Linux, this is needed in the config file `.irods/irods_environment.json` (be careful with commas to make sure it stays valid JSON):

```
"irods_plugins_home":  "/work/modules/Ubuntu/14.04/amd64/common/irods/4.1.9/var/lib/irods/plugins/"
```

On Triton, the corresponding directory is `"/share/apps/irods/4.1.9/var/lib/irods/plugins/"`
Practical usage

To be added once we have more specific use cases which are not covered above.

More resources

Documentation

- CSC webinar on IDA and opening data (2017): [https://www.youtube.com/watch?v=b8nVRgUBH0Q](https://www.youtube.com/watch?v=b8nVRgUBH0Q), [https://www.csc.fi/web/training/-/webinar_ida_2017](https://www.csc.fi/web/training/-/webinar_ida_2017).
- The CSC archive also uses irods, but it uses version 3 which is not compatible with these command line tools.

5.1.9 External links

- [Finland Open Science Initiative (ATT)](https://www.ottama.net/)

Cheatsheets: Data, A4 Data management plan.
Triton is the Aalto high performance computing cluster. It is your go-to resources for anything that exceeds your desktop computer’s capacity.

6.1 Triton cluster

Triton is the Aalto high-performance computing cluster. It serves all researchers of Aalto, but is currently coordinated from within the School of Science. Access is free for researchers (students not doing research should check out our intro for students). It is similar to the CSC clusters, though CSC clusters are larger and Triton is easier to use because it is more integrated into the Aalto environment.
6.1.1 Quick contents and links

**Triton contents**
- About Triton
  - Cluster overview
  - Usage policies and legal
  - Acknowledging Triton
- Getting Help/Contact
  - Triton issue tracker (most requests here, log in with HAKA)
  - Suggestions for good support requests
- Quick Reference
- Tutorials (start here)
  - Triton accounts
  - About Science-IT and Triton
  - Connecting to Triton
  - Applications
  - Software modules
  - Data storage
  - Interactive jobs
  - Serial Jobs
  - Array jobs
  - Job dependencies
  - Monitoring job progress and job efficiency
  - GPU computing
  - Parallel computing
- Cluster usage details
  - Parallel jobs (coming, for now see Running programs on Triton)
  - GPU Computing
- Applications

For full contents, see below.

**News Shortcuts**
- Issue tracker
- Quick Reference
- Triton Cheatsheet
- Triton FAQ
- Scicomp Garage

**Scientific computing resources**
- SCIP – Scientific Computing in Practice courses: organized by SciComp. Including Triton kick-starts and many others
- Parallel computing
- Aalto IT Services for Research
- Hands-on Scientific Computing: map of important computing skills
- Software Carpentry (scientific computation basics) and Code Refinery (more focused on programming techniques)

**General links**
- CSC - Finland’s academic computing center.
- FGCI user’s guide at CSC: That is a general Guide to FGCI resources. Triton is one of them.
- CSC HPC guides at CSC: a Triton like cluster at CSC. Similar setup, thus examples and instructions can be useful.
- Aalto research data management information

6.1.2 Overview

Cluster overview

Shared resource

Triton is a joint installation by a number of Aalto School of Science faculties within Science-IT project, which was founded in 2009 to facilitate the HPC Infrastructure in all of School of Science. It is now available to all Aalto researchers.

As of 2016, Triton is part of FGCI - Finnish Grid and Cloud Infrastructure (predecessor of Finnish Grid Infrastructure). Through the national grid and cloud infrastructure, Triton also becomes part of the European Grid Infrastructure.
## Hardware

<table>
<thead>
<tr>
<th>Node name</th>
<th>Number of nodes</th>
<th>Node type</th>
<th>Year</th>
<th>Arch (constraint)</th>
<th>CPU type</th>
<th>Memory Configuration</th>
<th>Infini-band</th>
<th>GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivb[1-24]</td>
<td>24</td>
<td>ProLiant SL230s G8</td>
<td></td>
<td>ivb avx</td>
<td>2x10 core Xeon E5 2680 v2 2.80GHz</td>
<td>256GB DDR3-1667</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>ivb[25-48]</td>
<td>24</td>
<td>ProLiant SL230s G8</td>
<td></td>
<td>ivb avx</td>
<td>2x10 core Xeon E5 2680 v2 2.80GHz</td>
<td>64GB DDR3-1667</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>pe[1-48,65-81]</td>
<td>65</td>
<td>Dell PowerEdge C4130</td>
<td>2016</td>
<td>hsw avx avx2</td>
<td>2x12 core Xeon E5 2680 v3 2.50GHz</td>
<td>128GB DDR4-2133</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>pe[49-64,82]</td>
<td>17</td>
<td>Dell PowerEdge C4130</td>
<td>2016</td>
<td>hsw avx avx2</td>
<td>2x12 core Xeon E5 2680 v3 2.50GHz</td>
<td>256GB DDR4-2133</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>pe[83-91]</td>
<td>8</td>
<td>Dell PowerEdge C4130</td>
<td>2017</td>
<td>bdw avx avx2</td>
<td>2x14 core Xeon E5 2680 v4 2.40GHz</td>
<td>128GB DDR4-2400</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>c[579-628,639-698]</td>
<td>110</td>
<td>ProLiant XL230a Gen9</td>
<td>2017</td>
<td>hsw avx avx2</td>
<td>2x12 core Xeon E5 2690 v3 2.60GHz</td>
<td>128GB DDR4-2966</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>c[629-638]</td>
<td>10</td>
<td>ProLiant XL230a Gen9</td>
<td>2017</td>
<td>hsw avx avx2</td>
<td>2x12 core Xeon E5 2690 v3 2.60GHz</td>
<td>256GB DDR4-2400</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>skl[1-48]</td>
<td>48</td>
<td>Dell PowerEdge C6420</td>
<td>2019</td>
<td>skl avx avx2 avx512</td>
<td>2x20 core Xeon Gold 6148 2.40GHz</td>
<td>192GB DDR4-2667</td>
<td>EDR</td>
<td></td>
</tr>
<tr>
<td>csl[1-48]</td>
<td>48</td>
<td>Dell PowerEdge C6420</td>
<td>2020</td>
<td>csl avx avx2 avx512</td>
<td>2x20 core Xeon Gold 6248 2.50GHz</td>
<td>192GB DDR4-2667</td>
<td>EDR</td>
<td></td>
</tr>
<tr>
<td>fn3</td>
<td>1</td>
<td>Dell PowerEdge R940</td>
<td>2020</td>
<td>avx avx2 avx512</td>
<td>4x20 core Xeon Gold 6148 2.40GHz</td>
<td>2TB DDR4-2666</td>
<td>EDR</td>
<td></td>
</tr>
<tr>
<td>gpu[1-10]</td>
<td>10</td>
<td>Dell PowerEdge C4140</td>
<td>2020</td>
<td>skl avx avx2 avx512 volta</td>
<td>2x8 core Intel Xeon Gold 6134 @ 3.2GHz</td>
<td>384GB DDR4-2667</td>
<td>EDR</td>
<td>4x V100 32GB</td>
</tr>
<tr>
<td>gpu[20-22]</td>
<td>3</td>
<td>Dell PowerEdge C4130</td>
<td>2016</td>
<td>hsw avx avx2 kepler</td>
<td>2x6 core Xeon E5 2620 v3 2.50GHz</td>
<td>128GB DDR4-2133</td>
<td>EDR</td>
<td>4x2 GPU K80</td>
</tr>
<tr>
<td>gpu[23-27]</td>
<td>5</td>
<td>Dell PowerEdge C4130</td>
<td>2017</td>
<td>hsw avx avx2 pascal</td>
<td>2x12 core Xeon E5 2680 v3 @ 2.5GHz</td>
<td>256GB DDR4-2400</td>
<td>EDR</td>
<td>4x P100</td>
</tr>
<tr>
<td>dgx[01-02]</td>
<td>2</td>
<td>Nvidia DGX-1</td>
<td>2018</td>
<td>bdw avx avx2 volta</td>
<td>2x20 core Xeon E5-2698 v4 @ 2.2GHz</td>
<td>512GB DDR4-2133</td>
<td>EDR</td>
<td>8x V100</td>
</tr>
<tr>
<td>gpu[28-37]</td>
<td>10</td>
<td>Dell PowerEdge C4140</td>
<td>2019</td>
<td>skl avx avx2 avx512 volta</td>
<td>2x8 core Intel Xeon Gold 6134 @ 3.2GHz</td>
<td>384GB DDR4-2667</td>
<td>EDR</td>
<td>4x V100 32GB</td>
</tr>
</tbody>
</table>

### 6.1. Triton cluster
All Triton computing nodes are identical in respect to software and access to common file system. Each node has its own unique host name and ip-address.

Networking

The cluster has two internal networks: Infiniband for MPI and Lustre filesystem and Gigabit Ethernet for everything else like NFS /home directories and ssh.

The internal networks are unaccessible from outside. Only the login node triton.aalto.fi has an extra Ethernet connection to outside.

High performance InfiniBand has fat-tree configuration in general. Triton has several InfiniBand segments (often called islands) distinguished based on the CPU arch. The nodes within those islands connected with different ratio like 2:1, 4:1 or 8:1, (i.e. in 4:1 case for each 4 downlinks there is 1 uplink to spine switches. The islands are ivb[1-45] 540 cores, pe[3-91] 2152 cores (keep in mind that pe[83-91] have 28 cores per node), four c[xxx-xxx] segments with 600 cores each, skl[1-48] and csl[1-48] with 1920 cores each [CHECKME]. Uplinks from those islands are mainly used for Lustre communication. Running MPI jobs possible on the entire island or its segment, but not across the cluster.

Disk arrays

All compute nodes and front-end are connected to DDN SFA12k storage system: large disk arrays with the Lustre filesystem on top of it cross-mounted under /scratch directory. The system provides about 1.8PB of disk space available to end-user.

Software

The cluster is running open source software infrastructure: CentOS 7, with SLURM as the scheduler and batch system.

Getting Triton help

There are many ways to get help, and you should try them all. If you are just looking for the most important link, it is our issue tracker.

Whatever you do, these guidelines for making good support requests are very useful.

See also:

Are you just looking for a Triton account? See Triton accounts.

Give enough information

We get many requests for help which are too vague to give a useful response. So, when sending us a question, always answer these questions and you’ll get the fastest useful response:

• Has it ever worked? (If so, what has changed?)
• What are you trying to accomplish? (Your ultimate goal, not current technical obstacle.)
• What did you do? (Be specific enough to be reproducible - copy and paste exact commands you run, scripts, inputs, output messages, etc.)
• What do you need? Do you need a complete solution, pointers to get started, or should we say if it will take too long and we recommend you think of other solutions first?
If you don’t know something, it’s OK, just do your best and we’ll help from there! You can also chat with us to brainstorm about issues in general. A much more detailed guide is available from Sigma2 documentation.

**The Triton docs**

In case you got to this page directly, you are now on the Triton and Science-IT (CS, NBE, PHYS at least) documentation site. See the main page for the index.

**Your colleagues**

Science is a collaborative process, even if it doesn’t seem so. Academic courses don’t teach you everything you need to know, so it’s worth trying to work together and learn from each other - your group is the expert in it’s work, after all.

**Issue tracker**

We keep track of cluster issues at https://version.aalto.fi/gitlab/AaltoScienceIT/triton/issues. Feel free to post your issue there. Either admins or other users can reply — and you should feel free to reply and help others, too. The system is accessible from anywhere in the world, but you need to login with HAKA (using the button). All newly created issues are reported to admins by email.

This is primary support channel and meant for general issues like general help, troubleshooting, problems with code, new software requests, problems that may affect several users.

**Note:** If you get a message that you are blocked from version.aalto.fi, send the email to servicedesk. It’s not your fault: it automatically blocks people when their organizational unit changes. Yes, this is bad but it’s not in our control. . .

If you have an Aalto visitor account, login with HAKA won’t work - use your email address and Aalto password.

**Email ticketing system**

For private issues you can also contact us via our email alias (on our wiki pages, login required). This is primarily intended for specific issues such as requesting new accounts, quotas, etc. Please avoid sending personal mails directly to admins, because it is best for all admins to be aware of issues, people may be absent, and personal emails are likely to be lost.

Most general issues should be reported to the issue tracker instead, not by email. Email is primarily for accounts related queries.

**Users’ mailing list**

All cluster users are on the triton-users mailing list (automagically kept in sync with those who have Triton access). It is for announcements and open discussions mainly, for problem solving please try the tracker.

If you do not receive list emails, you’d better check out with your local Triton admin that you are on the list. Otherwise you miss all the announcements including critical ones about maintenance breaks.
In person

Come by one of the Scientific computing garages any week. It’s the best place to get problems solved fast.

You can also come and talk to us face-to-face, but of course we have to be in-office. This is especially useful when there is an open-ended question where we have to discuss what is the best solution. We may then ask you to open a ticket once there is an answer, so that we can track the progress and not forget.

Triton support team

Most of us are members of your department’s support teams, so can answer questions about balancing use of Triton and your department’s computers. We also like it when people drop by and talk with us, so that we can better plan our services. In general, don’t mail us directly - use either the issue tracker above or the support email address. You can address your request to a specific person.

<table>
<thead>
<tr>
<th>Dept</th>
<th>Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS/NBE</td>
<td>Mikko Hakala</td>
<td>T-building A243 / Otakaari 3, F354</td>
</tr>
<tr>
<td>CS</td>
<td>Simo Tuomisto</td>
<td>T-building A243</td>
</tr>
<tr>
<td>PHYS</td>
<td>Simppa Akäslompolo</td>
<td>Otakaari 1, Y415a</td>
</tr>
<tr>
<td>PHYS</td>
<td>Ivan Degtyarenko</td>
<td>Otakaari 1, Y415a</td>
</tr>
<tr>
<td>CS/SCI</td>
<td>Richard Darst</td>
<td>T-building A243</td>
</tr>
<tr>
<td>NBE</td>
<td>Enrico Glerean</td>
<td>Otakaari 3, F354</td>
</tr>
</tbody>
</table>

Science-IT trainings

We have regular training in topics relevant to HPC and scientific computing. In particular, each January and June we have a “kickstart” course which teaches you everything you need to know to do HPC work. Each Triton user should come to one of these. For the schedule, see our training page.

Getting a detailed bug report with triton-record-environment

We have a script named triton-record-environment which will record key environment variables, input, and output. This greatly helps in debugging.

To use it to run a single command that gives an error:

```
triton-record-environment YOUR_COMMAND
Saving output to record-environment.out.txt
...
```

Then, just check the output of record-environment.out.txt (it shouldn’t have any confidential information, but make sure) and send it to us/attach it to the bug report.

If you use Python, add the `-p` option, matlab should use `-m`, and graphical programs should use `-x` (these options have to go before the command you execute).
**Triton accounts**

You need to request Triton access separately, however, the account information (username, password, shell, etc) is shared with the Aalto account so there is not actually a separate account. Triton access is available to any researcher at Aalto for free. Resources are funded by departments, and distributed by a fairshare algorithm: members of departments and schools which provide direct funding have a greater share.

Please use the account request form (“Triton: New user account”) to request the account. (For future help, you should probably use our issue tracker: see the [Getting Triton help](#) page.)

A few prerequisites:

- You must have valid Aalto account
- You must accept [Triton usage policies](#), including the data and privacy policies.
- Also tell us your department/school in your account creation request.
- You should have enough background to use Triton well, including Linux skills. Read [hands-on scientific computing](#), and you should know A (“Basics”), C (“Linux”), and D (“HPC”) well. Also see the [Triton tutorials](#).

Accounts are for:

- Researchers (as in, affiliated with a research PI in any way). Please tell us who your supervisor is in your account request.
- Students coming to one of our [Scientific Computing in Practice courses](#) which uses Triton. You will be specifically told if this is the case
- Other students not doing research needing computational facilities should check out our [introduction for students](#). This includes most student projects as part of courses, unless you are effectively joining a research group to do a project.

You know that you have Triton access if you are in the triton-users group at Aalto: [groups](#) shows this on Aalto linux machines.

**Your department/unit**

When you get an account, you get added to a unit’s group, which is “billed” for your usage. If you change Aalto units, this may need updated. Check `sshare -U` or `sshare` and if it’s wrong, let us know (the units are first on the line).

(These are currently by department, so changes are not that frequent)

**Password change and other issues**

Since your Triton account is a regular Aalto account, for any password change, shell change etc use Aalto services. You can always do these on the server kosh.aalto.fi (at least).

If you are in doubts, in case of any account related issue your primary point of contact is your local support team member via the support email address. Do not post such issues on the tracker.
Account deactivation / remove from mailing list

Your account currently lasts as long as your Aalto account does. If you want to end your account early, contact your local support via the scicomp email address. This way, you will also be unsubscribed from the triton-users maillist (the mailing list is required for everyone who has an account).

Before you leave, please clean up your home/work/scratch directories data. Consider who should have your data after you are done: does your group still need access to it? You won’t have access to the files after your account is deactivated. Note that scratch/work directory data are unrecoverable after deleting, which will happen eventually.

Terms of use/privacy policy

See the Usage policies and legal page.

Usage policies and legal

Acceptable Use Policy and Terms of Service

By using the Triton cluster resources, you shall be deemed to accept these conditions of use:

1. You shall only use Triton cluster to perform work, or transmit or store data consistent with the stated goals and policies of Aalto University and in compliance with these conditions of use.

2. You shall not use Triton cluster for any unlawful purpose and not (attempt to) breach or circumvent any administrative or security controls. You shall respect copyright and confidentiality agreements and protect your credentials (e.g. user login name, password, ssh private key), sensitive data and files.

3. You shall immediately report any known or suspected security breach or misuse of Triton cluster or credentials to the cluster support team.

4. Use of the cluster is at your own risk. There is no guarantee that the cluster will be available at any time or that it will suit any purpose.

5. Logged information, including information provided by you for registration purposes, shall be used for administrative, operational, accounting, monitoring and security purposes only in accordance with the policy below. This information may be disclosed to other organizations anywhere in the world for these purposes in the extent allowed by local laws. Although efforts are made to maintain confidentiality, no guarantees are given.

6. The cluster support team is entitled to regulate and terminate access for administrative, operational and security purposes and you shall immediately comply with their instructions.

7. You are liable for the consequences of any violation by you of these conditions of use.

8. You agree to explicitly mention and acknowledge the use of Science-IT resources in your work in any reports, workshops, papers or similar that result from such usage. Appropriate reference can be found at Acknowledgement of Triton usage.
Triton data (privacy) policy

Triton is a part of Aalto University IT systems, thus is fundamentally governed by the Aalto Privacy Policy for Employees or Privacy Policy for Students, the latest versions of which can always be found on aalto.fi.

For clarity, in this section, we describe the special cases of Triton data:

In summary:

- **The Triton account** is not a separate account, it is part of the Aalto account. We do not control that.

- **Triton usage statistics and logs.** Triton is used for university academic research only, so this information may be used for reporting and management in any way. Identifying information won’t be public, but note that it will be used for internal operations and contacting users as needed.

- **Data stored on Triton.** We are not the controller of this data. Data in your personal directories is controlled by you, and data in shared directories is controlled by the manager of that group. See the section below for more information on this data.

- **HAKA login data** (JupyterHub only). This is used to secure access to JupyterHub. Only your Aalto account name is requested, it is compared and immediately discarded (Triton is already linked to your Aalto account).

- **The triton-users mailing list** is automatically formed from all Aalto accounts in the triton-users group (everyone with an account). This is used to send service announcement and information related to scientific computing. This subscription is intrinsically tied to the Triton account and a requirement of the cluster usage. (Email information held by Aalto IT services).

We do not consider the Triton management data to consist of a personal data file (this is covered under Aalto policies), but for full disclosure we describe our use of data.

*Note about research data:* This section does not cover any data which users store on the cluster: for that, the user is the controller and Science-IT is only a processor. You are responsible for any administrative privacy matters. The following subsections relate only to administrative metadata.

**Controller and contact**

*Controller:* Aalto Science-IT, Aalto University, Espoo, Finland. *Contact information.* Please use the support email alias for account and personal data queries.

Account information comes from Aalto ITS registers.

**The purpose for processing the personal data**

Data is processed and stored in accordance with our agreement to provide a HPC cluster service including accounting and reporting, in accordance with the usage agreement. The cluster may only be used to support Aalto (not personal) activities, and all thus usage metadata represents Aalto activities and is owned by Aalto University.
Types of data

Triton stores the information necessary for provision of its services, including accounting, funding, and security. This includes logs of all operations and metadata of stored data. Data is only generated when a user uses the cluster. For example (including but not limited to):

- Connection logs
- Job submission and statistics logs
- Filesystem and storage metadata and logs

Uses of data

Data is used in the provision of the HPC cluster service. Primarily, this is through accounting, reporting, and scheduling of tasks. Historical data will automatically adjust future cluster priority.

Sources of information

Data is produced during the use of Triton for research purposes. This data is generated directly by users while using the cluster. Account information is provided by Aalto University, and in general not stored or processed here.

Data sharing

Data may be used for internal Aalto reporting and accounting (usually but not always aggregated at least at the group level), and used in non-identifiable forms in public reports and statistics. It may also be used as needed to investigate usage matters.

All users of the cluster may inspect the usage information and job statistics of the entire cluster (including all other users).

Timeframe

Data related to usage remains as long as the user has an active Triton account. Technical logging data allows accounting and reporting, and may be kept as long as needed for security and reporting purposes (indefinitely). Where possible, this may be in anonymous form.

Legal notices

Data is stored in Finland in Aalto or CSC approved facilities. Access is only via Aalto account.

You may request rectification of your data. However, most data is technical logging information which can not be removed or changed.

You may cease using the cluster, remove your research data, and request your account be closed (this does not close your Aalto account because we do not control that), but historical usage data will remain for accounting purposes. Should technical errors in data be identified, a bug should be reported.

You may access and extract your own data using the standard interfaces described in the user guide.

Identifiable administrative metadata and accounting data is not transferred outside of the EU/EEA except under proper agreement. (We have to say that, but in reality identifiable data is never transferred out of Aalto or maybe the FGCI consortium in Finland).
You may lodge a complaint with the Aalto data protection officer (see Aalto privacy notices for up to date contact information) or the Finnish supervision authority Tietosuoja.

**Research and home data stored on cluster**

We provide a storage service for for data stored on the cluster (scratch and home directories):

Our responsibility is limited to keeping this data secure and providing access to the corresponding Aalto accounts. The shared directory manager should be able to make choices about data. We do not access this data except with an explicit request, but for management purpose we do use the file metadata\((\text{stat } $filename)\). For full information, see the Science-IT data policy.

- We do not look into private files without your explicit request (if you want help with something, explicitly tell us if we can look at them).
- If your files are made cluster-readable (the \texttt{chmod} “other” permissions), you give permission for others to look at contents. Note that this is not the default setting.
- Should you report a problem, we may run \texttt{stat} as superuser on relevant files to determine basic metadata without further checks.
- Should you have a problem that requires us to look at the contents of files or directories, we must first have your explicit permission (either in writing or in person)
- User-owned data (home directories, work directories) may be deleted six months after an account expires. Use a group-based storage space instead.

Our data storage service is suitable for confidential data. You must ensure that permissions are such that technical access is limited.

**Acknowledging Triton**

**Acknowledgement line**

Triton and Science-IT gets funding from departments and Aalto, so it is critical that we show them the results of our work. Thus, please note that if you use the cluster for research that is published or presented in a talk or poster form you must acknowledge the Science-IT project by School of Science, that funds the Triton and affiliated resources. By published work we mean everything like articles, doctoral theses, diplomas, reports, other relevant publications. Use of triton can be anything: CPUs, GPUs, or the storage system (note that the storage system is the “scratch” system, which is cross-mounted to several different departments - you can use Triton without logging into it.)

An appropriate acknowledgement line might be one of:

- We acknowledge the computational resources provided by the Aalto Science-IT project.
- The calculations presented above were performed using computer resources within the Aalto University School of Science “Science-IT” project.

You can decide which one fits better to your text/slides. Rephrasing is also fine, the main issue is referencing to Science-IT and Aalto. (Note that this does not exist in various funding databases, this is an Aalto internal project.)
Reporting

We can’t automatically track all the Triton publications. We need all users to link the publications to Science-IT in ACRIS. It takes about 30 seconds if you aren’t looking at ACRIS now, or 5 when you are already there. All publications are required to be in ACRIS anyway, so this is a fast process.

You can see the already-reported publications here: https://research.aalto.fi/en/equipment/scienceit(27991559-92d9-4b3b-95ee-77147899d043)/publications.html

Instructions:

1. Log in to ACRIS: https://acris.aalto.fi
2. Find your publication: Research Output (left sidebar) -> Click on your publication
   • If your publication is not already there, then see your department’s ACRIS instructions, or the general help below.
3. Link it to Science-IT: scroll down to “Relations” -> “Facilities/Equipment” -> Search “Science-IT” and select it. (This is on the main page, not the separate “Relations” page.)
4. Click Save at the bottom of the window.
5. Repeat for all publications (and datasets, etc.)

You are done! You can see if your publications appears on the list above.

More help:

• General ACRIS help: ACRIShelp
• Manually adding journal article (most are automatically transferred): Submitting a journal article on ACRIShelp.

Should you have problems, first contact your department’s ACRIS help. If a publication or academic output somehow can’t be linked, let us know and we will make sure that we include it in our own lists.

6.1.3 Tutorials

These are designed to be read (or watched) in-order by every Triton user when they get their accounts (except maybe the last ones). In order to use Triton well, in the Hands-on SciComp roadmap you should also know the Basics (A) and Linux (C) levels as a prerequisite.

About these tutorials

Video

Watch this in the Winter Kickstart 2021 course.

The “Shell crash course” is a useful prerequisite. You can watch a short version (20 min) or longer version (1 hour).

Or see the full playlist.

Welcome to the Aalto Scientific Computing High-performance computing (HPC) tutorials. These tutorials will get you started with the Triton cluster.

Despite the HPC in the name, most of these tutorials are not about the high-performance part: instead, we get you started using and submitting jobs to the cluster. These days, many people use a cluster for simple jobs: getting more stuff done at once, not a few very big tasks. Doing the big tasks are a more specialized topic, which this will introduce you to and you will be able to use other software for that. Programming your own HPC software is out of our scope.
6.1. Triton cluster
Not at Aalto?

These tutorials use Aalto’s cluster as an example, but they are designed to be as general as possible. You can probably follow along and learn useful topics here anyway, but you will have. We recommend you review the basics of your cluster’s documentation first, then read this. You will have to translate somethings to your own specifics, but hopefully our examples can inspire you to understand your own docs better.

These tutorials will be quite useful if you have:

- A cluster using Slurm as the batch system
- You can get a shell on that server
- The git installed
- Python 2 or 3 installed as python

Unfortunately, not all clusters are standardized (though probably they should be slightly more than they are). Things that may be different (we’ll try to point these out where relevant):

- The way you connect to the cluster, including remote access methods.
- Exact names of batch partitions.
- The slurm utility probably isn’t installed, seff may not be there.
- Module names for software.
- You probably don’t have our singularity stuff installed.
- Parallel and GPU stuff is probably different.

Running another cluster?

If you run your own cluster, create a quick reference such as Triton quick reference so that others following tutorials such as this can quickly translate to your own cluster’s specifics.

Don’t you think clusters should be more interoperable? It’s something we are thinking about.

What’s next?

Introduce yourself to the cluster resources at Aalto.

About Science-IT and Triton

Video

Watch this in the Winter Kickstart 2021 course

This is the first tutorial. The next is Connecting to Triton.

Science-IT is an Aalto infrastructure for scientific computing. Its roots was a collaboration between the Information and Computer Science department (now part of CS), Biomedical Engineering and Computational Science department (now NBE), and Applied Physics department. Now, it still serves all Aalto and is organized from the School of Science.

You are now at the first step of the Triton tutorial.
About Triton

Triton is a mid-sized heterogeneous computational Linux cluster. This means that we are not at a massive scale (though we are, after CSC, the largest publically known known cluster in Finland). We are heterogeneous, so we continually add new hardware and incrementally upgrade. We are designed for scientific computing and data analysis. We use Linux as an operating system (like most supercomputers). We are a cluster: many connected nodes with a scheduling system to divide work between them. The network and some storage is shared, CPUs, memory, and other storage is not shared.

A real Ship of Theseus

In the Ship of Theseus thought experiment, every piece of a ship is incrementally replaced. Is it the same ship or not?

Triton is a literal Ship of Theseus. Over the ~10 years it has existed, every part has been upgraded and replaced, except possibly some random cables and other small parts. Yet, it is still Triton. Most clusters are recycled after a certain lifetime and replaced with a new one.

On an international scale of universities, the power of Triton is relatively high and it has a very diverse range of uses, though CSC has much more. Using this power requires more effort than using your own computer - you will need to get/be comfortable in the shell, using shell scripting, managing software, managing data, and so on. Triton is a good system to use for learning.

Getting skills

See also:

Main article: Training

As time goes on, computers are getting easier and easier to use. However, research is not a consumer product, and the fact is that you need more knowledge to use Triton than most people learn in academic courses.

Science-IT has created a (still under development) modular training plan, which divides useful knowledge into levels. In order to use Triton well, you need to be somewhat proficient at Linux usage (C level). In order to do parallel work, you need to be good at the D-level and also somewhat proficient at the HPC level (E-level). This tutorial and user guide covers the D-level, but it is up to you to reach the C-level first.

See our training program and plan for suggested material for self-study and lessons. We offer routine training, see our Scientific Computing in Practice lecture series page for info.

Getting help

See also:

Main article: Getting Triton help

There are many ways to get help. Most daily questions should go to our issue tracker (direct link), which is hosted on Aalto Gitlab (login with the HAKA button). This is especially important because many people end up asking the same questions, and in order to scale everyone needs to work together.

Please, don’t send us personal email, because not everyone is here all the time and you may end up asking someone other than the best person. Personal email is also very likely to get lost. By the same token, we have a service email address, but this should only be used for account matters. If it affects others (software, usage problems, etc), use the issue tracker, otherwise we will point you there and spend lots of time answering the same questions over and over.

Also, always search this scicomp docs site and old issues in the issue tracker.

6.1. Triton cluster
We have weekly “SciComp garage” sessions where we provide help in person.

However, the most important thing is to be able to continually develop your skills to help yourself and your colleagues. See the previous section for our solution for this.

Software

Triton, being a shared system, has more complicated software requirements. In an upcoming tutorial, you will learn how to use existing software. Be aware that installing your own is possible (and people do it all the time), but does require some attention to details. Either way, you will need to know the basics of software on Linux.

What's next?

The next tutorial is Connecting to Triton.

Connecting to Triton

Video

Watch this in the Winter Kickstart 2021 course

Prerequisites

The shell crash course is a prerequisite to this material (and the tutorial in general).

Summary:

You can connect to triton.aalto.fi via ssh from Aalto and CSC networks. Aalto networks include: Wired workstation networks, eduroam, and the aalto wireless network only if you are using an Aalto managed laptop (otherwise aalto is like aalto open). If you connect to the Aalto VPN, you will be on the Aalto networks.

For SSHing to Triton from outside of your department or CSC, please login first to a university server (like kosh.aalto.fi or taltta.org.aalto.fi) and then open a session to triton.aalto.fi - or use the –J option in modern ssh.

Important: Triton uses Aalto accounts, but your account must be activated first.

Note: Are you here for a SciComp KickStart course? You just need to make sure you have an account and then be able to connect via ssh (first section here), and you don’t need to worry about the graphical application parts. Everything else, we do tomorrow.
The way you connect will be different in every site, but you should be able to get a terminal somehow.

There are different ways of connecting:

<table>
<thead>
<tr>
<th>Method</th>
<th>About</th>
<th>From where?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssh</td>
<td>Works everywhere, from everywhere. Firewalls may make things hard sometimes.</td>
<td>Aalto networks only, otherwise ssh to kosh and then Triton</td>
</tr>
<tr>
<td><a href="https://jupyter.triton.aalto.fi">https://jupyter.triton.aalto.fi</a></td>
<td>Jupyter interface, but provides shell access via web browser.</td>
<td>Whole internet</td>
</tr>
<tr>
<td><a href="https://vdi.aalto.fi">https://vdi.aalto.fi</a></td>
<td>Virtual desktop, from there you have to ssh to Triton anyway but gets you past firewalls and can run graphical programs via SSH.</td>
<td>Whole internet</td>
</tr>
</tbody>
</table>

**Connecting via ssh**

**Linux**

All Linux distributions come with an `ssh` client, so you don’t need to do anything. To use graphical applications, use the standard `–X` option, nothing extra needed.:

```bash
ssh triton.aalto.fi
# OR, if your username is different: 
ssh username@triton.aalto.fi
```

If you are from outside the Aalto networks, use the ProxyJump option in modern OpenSSH:

```bash
ssh -J kosh.aalto.fi triton.aalto.fi
# OR, if your username is different: 
ssh -J username@kosh.aalto.fi username@triton.aalto.fi

# If you do not have the -J option: 
ssh kosh.aalto.fi
ssh triton.aalto.fi
```

**Mac**

`ssh` is installed by default, same as Linux. Run it from a terminal, same command as Linux. To run graphical applications, you need to install an X server (XQuartz).

**Windows**

You need to install a ssh client yourself: PuTTY is the standard one. If you want to run graphical programs, you need an X server on Windows: see this link for some hints. (Side note: putty dot org is an advertisement site trying to get you to install something else.)

You should configure this with the hostname, username, and save the settings so that you can connect quickly.

If you are outside the Aalto networks, you need to first connect to kosh.aalto.fi or some other server, and then use the Linux instructions to connect to Triton.
Advanced options

You can verify the ssh key fingerprints.

See the advanced ssh information to learn how to log in without a password, automatically save your username and more. It really will save you time.

ssh is one of the most fundamental Linux programs: by using it well, you can really do almost anything from anywhere. The .ssh/config file is valuable to set up. If ssh is annoying to use, ask for some help in getting it working well.

Exercise

1. Connect to Triton. List your home directory and work directory $WRKDIR.
2. Check the uptime and load of the login node: uptime and htop (q to quit - if htop is not available, then top works almost as well). What else can you learn about the node?
3. Check what your default shell is: echo $SHELL. Go ahead and change your shell to bash if it’s not yet (see below).

Change your shell to bash (Aalto)

Only needed if your shell isn’t already bash. If echo bash reports /bin/bash, then you are already using bash.

The thing you are interacting with when you type is the shell - the layer around the operating system. bash is the most common shell, but the Aalto default shell used to be zsh (which is more powerful in some ways, but harder to teach with). If you joined Aalto after autumn 2018, you probably don’t need to do anything. We recommend that you check and change your shell to bash.

You can determine if your shell is bash by running echo $SHELL. Does it say /bin/bash?

If not, ssh to kosh.aalto.fi and run chsh -s /bin/bash. It may take 15 minutes to update, and you will need to log in again.

Connecting via https://jupyter.triton.aalto.fi

Jupyter is a web-based way of doing computing. But what some people forget is that it has a full-featured terminal and console included.

Go to https://jupyter.triton.aalto.fi (not .cs.aaalto) and log in. Select “Slurm 5 day, 2G” and start.

To start a terminal, click File→New→Terminal - you do anything you need to do from here, same as ssh. If you need to edit text files, you can also do that through JupyterLab (note: change to the right directory before creating a new file!).

To learn more about Jupyterlab, you need to read up elsewhere, there are plenty of tutorials.
Connecting via https://vdi.aalto.fi

If you go to https://vdi.aalto.fi, you can access a cloud-based Aalto workstation. HTML access works from everywhere, or download the “VMWare Horizon Client” for a better connection. Start a Ubuntu desktop (you get Aalto Ubuntu). From there, you have to use the normal ssh instructions (via the Terminal application) using the instructions you see above: ssh triton.aalto.fi.

For more information, see the IT help.

What’s next?

The next tutorial is about software availability in general.

Applications

Video

Watch this in the Winter Kickstart 2021 course

In this tutorial, we talk about the overall process of finding, building, and compiling software. These days, installing and managing scientific software is taking more and more time, thus we need to specifically talk about it.

See also:

Main article: Applications: General info

Local differences

Almost every site will use modules. The exact module names, and anything beyond that, will be different. Containers are becoming more common, but they are less standardized.

How to find the software you need

You can find what softwares we have available in different ways:

- First, you should check our Applications page and see if the software you need is already available and has instructions.

- If you find the software you need available, you can usually load it via a module. The next tutorial, Software modules explains what modules are and how to work with them.

- You can also search this tutorial to see what you can find (though note that not everything is in the Triton section here - some applies to Aalto workstations or own computers).

- It’s always a good idea to search the issue tracker to see if there are previous issues about it - not everything is always updated.

- Ask other users in the Zulip chat. We hope that we can facilitate user group meetings and discussion among users of similar software suites.

- Ask us admins in garage.

Throughout this process, try to remember these things:
1. Scientific software, like scientific process itself, is collaborative. Work on sharing and seeking knowledge among other users. They might have the answer you need.

2. Interesting problems draw people together independently. If you’re working on a certain type of a problem, it is quite likely that some other researcher is working on a similar problem. You might not be alone with your problem.

3. Try to form connections between users of similar software. The same software that you use can be used by a researcher in completely different field. Many software suites e.g. statistical modelling, machine learning, is common to many other fields. If you cannot find similar users within your field, look across fields.

4. If you find something useful or interesting, share it. If you do not know who to share it with, share it with us in SciComp. When we hear of a tool, a method, a success story or a problem encountered by one of our users, we often try to share it among other researchers.

**Common applications are available as modules**

**Important:** This is Aalto-specific. Some of these will work if you `module load fgci-common` at other Finnish sites (but not CSC). This is introduced in the next lesson.

Here is a sample of our most commonly used software:

- **Python:** `module load anaconda` for the Anaconda distribution of Python 3, including a lot of useful packages. [More info.](#)
- **R:** `module load r` for a basic R package. [More info.](#)
- **Matlab:** `module load matlab` for the latest Matlab version. [More info.](#)
- **Julia:** `module load julia` for the latest Julia version. [More info.](#)

If one of these `module load` commands does not work at your site, try `module spider $NAME` and see if you can find it. More information on these commands will be actually covered under the upcoming modules tutorial.

We try to install commonly used software for all of our users, so that everyone can benefit from them. If you cannot find what you’re looking for, do let us know.

**Singularity containers**

**See also:**

Main article: *Singularity Containers*

Some software packages are either very complicated to install or they have been designed with certain operating systems in mind. For these kinds of software we often use containers. A software container is basically a complete self-contained operating system environment. Another advantage of containers is that they make it easy to move installed software from system to system, so that you can have the same environment everywhere.

If your program is usually deployed using Docker or it is hard to maintain, do read our documentation on Singularity containers and contact us for more information.

We also provide *some containers built by NVIDIA*. These containers are from NVIDIA’s NGC-repository and meant for GPU computations.
Requesting new software

We aim to install a good base of software for our users - but it's not possible to keep up with all requests. If you need something, submit a request to our issue tracker, but be aware that despite best efforts, we can’t do everything. See the main Applications page for more information.

A plea: make your software reusable!

Five years from now, when you are releasing your own software that you want others to use, make it easy to install and reusable.

Exercises

If you are at Aalto, everything will work. Otherwise, if you are in Finland (but not at CSC) module load fgci-common will make our modules available on your cluster.

1. Figure out how to use tensorflow (this is not a software problem, but a searching the documentation problem). Make it work enough to do python and import tensorflow.

2. Find the Applications page link above, and check the list for ways to find out if we already have your software installed. See if we have what you need, using any of the strategies on that list.

3. (optional) From the Applications page, find the Spack package list (warning: it’s a very long page and takes a while to load). Does it have anything useful to you?

4. (optional) Discuss among your group what software you need, if it’s available, and how you might get it.

What’s next?

The next tutorial covers software modules in more detail.

Software modules

Video

Watch this in the Winter Kickstart 2021 course

There are hundreds of people using every cluster. They all have different software needs, including conflicting versions required for different projects! How do we handle this without making a mess, or one person breaking the cluster for everyone?

This is actually a very hard, but solved within certain parameters, problem. Software installation and management takes up a huge amount of our time, but we try to make it easy for our users. Still, it can end up taking a lot of your effort as well.

Local differences

Almost every site uses modules, and most use the same Lmod system we use here. But, the exact module names you can load will be different.
Introduction to modules

The answer is the standard “module” system Lmod. It allows us to have unlimited number of different software packages installed, and the user can select what they want. Modules include everything from compilers (+their required development files), libraries, and programs. If you need a program installed, we will put it in the module system.

In a system the size of Triton, it just isn’t possible to install all software by default for every user.

A module lets you adjust what software is available, and makes it easy to switch between different versions.

As an example, let’s inspect the gcc module (abbreviated output shown) with `module show gcc`:

```bash
$ module show gcc
---
whatis("Name : gcc")
whatis("Version : 9.2.0")
whatis("Short description : The GNU Compiler Collection includes front ends for C, C++, Objective-C, Fortran, Ada, and Go, as well as libraries for these languages.")
whatis("Configure options : --disable-multilib --enable-languages=c,c++,fortran --with-mpf=/share/apps/spack/envs/fgci-centos7-generic/software/mpfr/3.1.6/m6xmzs --with-gmp=/share/apps/spack/envs/fgci-centos7-generic/software/gmp/6.1.2/mnsg5g2 --enable-lto --with-quad --with-system-zlib --with-mpc=/share/apps/spack/envs/fgci-centos7-generic/software/mpc/1.1.0/uaijipe --with-isl=/share/apps/spack/envs/fgci-centos7-generic/software/isl/0.19/indu5p6")
help(["The GNU Compiler Collection includes front ends for C, C++, Objective-C, Fortran, Ada, and Go, as well as libraries for these languages."])
family("compiler")
prepend_path("PATH","/share/apps/spack/envs/fgci-centos7-generic/software/gcc/9.2.0/dnrscms/bin")
prepend_path("MANPATH","/share/apps/spack/envs/fgci-centos7-generic/software/gcc/9.2.0/dnrscms/share/man")
prepend_path("LIBRARY_PATH","/share/apps/spack/envs/fgci-centos7-generic/software/gcc/9.2.0/dnrscms/lib")
prepend_path("LD_LIBRARY_PATH","/share/apps/spack/envs/fgci-centos7-generic/software/gcc/9.2.0/dnrscms/lib")
prepend_path("LIBRARY_PATH","/share/apps/spack/envs/fgci-centos7-generic/software/gcc/9.2.0/dnrscms/lib64")
prepend_path("LD_LIBRARY_PATH","/share/apps/spack/envs/fgci-centos7-generic/software/gcc/9.2.0/dnrscms/lib64")
prepend_path("CPATH","/share/apps/spack/envs/fgci-centos7-generic/software/gcc/9.2.0/dnrscms/include")
prepend_path("CMAKE_PREFIX_PATH","/share/apps/spack/envs/fgci-centos7-generic/software/gcc/9.2.0/dnrscms/")
setenv("CC","/share/apps/spack/envs/fgci-centos7-generic/software/gcc/9.2.0/dnrscms/bin/gcc")
setenv("CXX","/share/apps/spack/envs/fgci-centos7-generic/software/gcc/9.2.0/dnrscms/bin/g++")
...
```

The command `module show gcc` shows some meta-info (name of the module, its version, etc.) When you load this module, it adjusts various environment paths (as you see there), so that when you type `gcc` it runs the program from `/share/apps/spack/envs/fgci-centos7-generic/software/gcc/9.2.0/dnrscms/bin/gcc`. This is almost magic: we can have many versions of any software installed, and everyone can pick what they want, with no conflicts.
Loading modules

Let’s dive right into an example and load a module.

Local differences

If you are not at Aalto, but in Finland (but not at CSC), then you need to run `module load fgci-common` first, before any of the other commands will work (and you will need to *keep doing this* for every other tutorial in this series). You have to do this *every time you start a new shell*. If you are at CSC or not in Finland, the concepts here also apply to you, but the actual names of the modules loaded may differ.

Let’s assume you’ve written a Python script that is only compatible with Python version 3.7.0 or higher. You open a shell to find out where and what version our Python is. The `which` program looks up the current detected version of a program - very useful when testing modules:

```bash
$ which python3
/usr/bin/python3
$ python3 -V
Python 3.6.8
```

But you need a newer version of Python. To this end, you can **load** the *anaconda* module using the `module load anaconda` command, that has a more up to date Python with lots of libraries already included:

```bash
$ module load anaconda
$ which python
/share/apps/anaconda-ci/fgci-centos7-generic/software/anaconda/2020-03-tf2/521551bc/˓
→ /bin/python
$ python -V
Python 3.6.10 :: Anaconda, Inc.
```

As you see, you now have a newer version of Python, in a different directory.

You can see a list of the all the loaded modules in our working shell using the `module list` command:

```bash
$ module list
Currently Loaded Modules:
  1) anaconda/2020-03-tf2
```

**Note:** The `module load` and `module list` commands can be abbreviated as `ml`

Let’s use the `module purge` command to **unload** all the loaded modules (*anaconda* in this case):

```bash
$ module purge
```

Or explicitly unload the *anaconda* module by using the `module unload anaconda` command:

```bash
$ module unload anaconda
```

You can load any number of modules in your open shell, your scripts, etc. You could load modules in your `~/.bash_profile`, but then it will always automatically load it - perhaps even if you don’t expect it. Watch out for this if you get un-explainable bugs - it may be best to explicitly load what you need.

6.1. Triton cluster
Type-along: Where is Matlab?

Let’s say you want to use Matlab. You log in and try in the shell:

```bash
$ matlab
-bash: matlab: command not found
```

So first search for it using the `module spider` command:

```bash
$ module spider matlab
```

```bash
----------------------------------------------------------------------------
matlab:                                                                
----------------------------------------------------------------------------
Versions:
  matlab/r2012a
  matlab/r2014a
  matlab/r2015b
  matlab/r2016a
  matlab/r2016b
  matlab/r2017b
  matlab/r2018a
  matlab/r2018b
  matlab/r2019a
  matlab/r2019b
```

We see there are a lot of versions available.

Load the latest version of Matlab as:

```bash
$ module load matlab
```

It never hurts to double check the version and in fact is recommended. So let’s do just that:

```bash
$ module list
Currently Loaded Modules:
  1) matlab/r2019b
```

Type-along: Where is R?

If you don’t specify the version - just as the above Matlab example - the default version of the module is usually loaded, which is usually the latest version. The default version, however, is not always latest version. To see an example, let’s see what versions of R are available:

```bash
$ module spider r
```

```bash
----------------------------------------------------------------------------
r:                                                                
----------------------------------------------------------------------------
Versions:
  r/3.4.3-python-2.7.14
  r/3.4.3-python2
  r/3.4.3-python3
  r/3.5.0-python-2.7.14
  r/3.5.0-python2
```

(continues on next page)
Let’s try loading the default version:

```bash
$ module load r
```

You can list all the dependencies the R module requires and loads:

```bash
$ module list
```

The last loaded module clearly shows that the version of the R loaded is `r/3.4.3-python2`. To load the latest version of R, use the **fullName** of the module:

```bash
$ module load r/3.6.1-python3
```

### Module versions

**What’s the difference between module load r and module load r/3.6.1-python3?**

The first loading `r` loads the version that Lmod assumes to be the latest one. **This is necessarily not the latest one.** The second loading `r/3.6.1-python3` loads that exact version, which is supposed to not change. If you’re not interested about the specific version, you can load it without the version (but then **when** stuff randomly stops working, you’re going to have to figure out what happened). Once you are past that (possibly from day one!), it’s usually a good idea to load specific version, so that your environment will stay the same until you are done.

This is most important for compiled software, but applies to everything.
What’s going on under the hood here?

In Linux systems, different environment variables like $PATH and $LD_LIBRARY_PATH help figure out how to run programs. Modules just cleverly manipulate these so that you can find the software you need, even if there are multiple versions available. You can see these variables with the echo command, e.g. echo $PATH.

When you load a module in a shell, the module command changes the current shell’s environment variables, and the environment variables are passed on to all the child processes.

You can explore more with module show $NAME.

Making a module collection

There is a basic dependency/conflict system to handle module dependency. Each time you load a module, it resolves all the dependencies. This can result in long loading times or be annoying to do each time you log in to the system. However, there is a solution: module save $collection_name and module restore $collection_name

Let’s see how to do this in an example.

Let’s say that for compiling / running your program you need:

- a compiler
- CMake
- MPI libraries
- FFTW libraries
- BLAS libraries

You could run this each time you want to compile/run your code:

```bash
$ module load gcc/9.2.0 cmake/3.15.3 openmpi/3.1.4 fftw/3.3.8-openmpi openblas/0.3.7
$ module list   # 15 modules
```

Let’s say this environment works for you. Now you can save it with module save my-env. Then module purge to unload everything. Now, do module restore my-env:

```bash
$ module save my-env
$ module purge
$ module restore my-env
$ module list   # same 15 modules
```

Generally, it is a good idea to save your modules as a collection to have your desired modules all set up each time you want to re-compile/re-build.

So the subsequent times that you want to compile/build, you simply module restore my-env and this way you can be sure you have the same previous environment.

**Note:** You may occasionally need to rebuild your collections in case we re-organize things (it will prompt you to rebuild your collection and you simply save it again).
Full reference

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module load NAME</td>
<td>load module</td>
</tr>
<tr>
<td>module avail</td>
<td>list all modules</td>
</tr>
<tr>
<td>module spider NAME</td>
<td>search modules</td>
</tr>
<tr>
<td>module list</td>
<td>list currently loaded modules</td>
</tr>
<tr>
<td>module show NAME</td>
<td>details on a module</td>
</tr>
<tr>
<td>module help NAME</td>
<td>details on a module</td>
</tr>
<tr>
<td>module unload NAME</td>
<td>unload a module</td>
</tr>
<tr>
<td>module save ALIAS</td>
<td>save module collection to this alias (saved in ~/.lmod.d/)</td>
</tr>
<tr>
<td>module restore ALIAS</td>
<td>load saved module collection (faster than loading individually)</td>
</tr>
<tr>
<td>module purge</td>
<td>unload all loaded modules (faster than unloading individually)</td>
</tr>
</tbody>
</table>

Final notes

If you have loaded modules when you build/install software, remember to load the same modules when you run the software (also in Slurm jobs). You’ll learn about running jobs later, but the `module load` should usually be put into the job script.

The modules used to compile and run a program become part of its environment and must always be loaded.

We use the Lmod system and Lmod works by changing environment variables. Thus, they must be `sourced` by a shell and are only transferred to child processes. Anything that clears the environment clears the loaded modules too. Module loading is done by special functions (not scripts) that are shell-specific and set environment variables.

Triton modules are also available on Aalto Linux: use `module load triton-modules` to make them available.

Some modules are provided by Aalto Science-IT, some by CSC. You could even make your own user modules.

Exercises

Before each exercise, run `module purge` to clear all modules.

1. `module avail` and check what you see. Find a software that has many different versions available. Load the oldest version.

2. `PATH` is an environment variable that shows from where programs are run. See its current value using `echo $PATH`. Then, load a module such as `py-gpaw`. List what it loaded. Check the value of `PATH` again. Why is there so much stuff?

3. (Advanced) Same as number 2, but use `env | sort > filename` to store environment variables, then `swap to py-gpaw/1.3.0-openmpi-scalapack-python3`. Do the same, and compare the two outputs using `diff`.

4. Load a module with many dependencies, such as `r-ggplot2` and save it as a collection. Compare the time needed to load the module and the collection. (Does `time` not work? Change your shell to bash, see the previous tutorial)

5. (Advanced) Load `openfoam-org/7-openmpi-metis`. Use `which` to find where executable `blockMesh` is coming from and then use `ldd` to find out what libraries it uses.
What's next?

The next tutorial covers data storage.

Data storage

Video

Watch this in the Winter Kickstart 2021 course

In this tutorial, we go over places to store data on Triton and how to access it remotely.

Optimizing data storage isn’t very glamorous, but is an important part of high-performance computing.

This page roughly has three parts:

• What storage locations are available?
• Where are they available (mounted) on Aalto computers?
• How can you access them remotely?

Basics

Triton has various ways to store data. Each has a purpose, and when you are dealing with large data sets or intensive I/O, efficiency becomes important.

Roughly, we have small home directories (only for configuration files), large Lustre (scratch and work, large, primary calculation data), and special places for scratch during computations (local disks). At Aalto, there is aalto home, project, and archive directories which, unlike Triton, are backed up but don’t scale to the size of Triton.

A file consists of its contents and metadata. The metadata is information like user, group, timestamps, permissions.

To view metadata, use ls -l or stat.

Filesystem performance can be measured by both IOPS (input-output operations per second) and stream I/O speed. /usr/bin/time -v can give you some hints here. You can see the profiling page for more information.

Think about I/O before you start! - General notes

When people think of computer speed, they usually think of CPU speed. But this is missing an important factor: How fast can data get to the CPU? In many cases, input/output (IO) is the true bottleneck and must be considered just as much as processor speed. In fact, modern computers and especially GPUs are so fast that it becomes very easy for a few GPUs with bad data access patterns to bring the cluster down for everyone.

The solution is similar to how you have to consider memory: There are different types of filesystems with different trade-offs between speed, size, and performance, and you have to use the right one for the right job. Often times. So you have to use several in tandem: For example, store original data on archive, put your working copy on scratch, and maybe even make a per-calculation copy on local disks. Check out wikipedia:Memory Hierarchy and wikipedia:List of interface bit rates.

The following factors are useful to consider:

• How much I/O are you doing in the first place? Do you continually re-read the same data?
• What’s the pattern of your I/O and which filesystem is best for it? If you read all at once, scratch is fine. But if there are many small files or random access, local disks may help.
• Do you write log files/checkpoints more often than is needed?
• Some programs use local disk as swap-space. Only turn on if you know it is reasonable.

There’s a checklist in the storage details page.
Avoid many small files! Use a few big ones instead. (we have a dedicated page on the matter)

Available data storage options

Each storage location has different sizes, speed, types of backups, and availability. You need to balance between these.

<table>
<thead>
<tr>
<th>Name</th>
<th>Path</th>
<th>Quota</th>
<th>Backups</th>
<th>Locality</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
<td>$HOME or /home/$username/</td>
<td>hard quota 10GB</td>
<td>Night</td>
<td>all nodes</td>
<td>Small user specific files, no calculation data.</td>
</tr>
<tr>
<td>Work</td>
<td>$WRKDIR or /scratch/work/$username/</td>
<td>200GB and 1 million files</td>
<td>x</td>
<td>all nodes</td>
<td>Personal working space for every user. Calculation data etc. Quota can be increased on request.</td>
</tr>
<tr>
<td>Scratch</td>
<td>/scratch/$dept/$project/</td>
<td>on request</td>
<td>x</td>
<td>all nodes</td>
<td>Department/group specific project directories.</td>
</tr>
<tr>
<td>Local temp</td>
<td>/tmp/</td>
<td>limited by disk size</td>
<td>x</td>
<td>single-node</td>
<td>Primary (and usually fastest) place for single-node calculation data. Removed once user’s jobs are finished on the node.</td>
</tr>
<tr>
<td>Local persist.</td>
<td>/l/</td>
<td>varies</td>
<td>x</td>
<td>dedicated group servers only</td>
<td>Local disk persistent storage. On servers purchased for a specific group. Not backed up.</td>
</tr>
<tr>
<td>ramfs</td>
<td>$XDG_RUNTIME_DIR</td>
<td>limited by memory</td>
<td>x</td>
<td>single-node</td>
<td>Ramfs on the login node only, in-memory filesystem</td>
</tr>
</tbody>
</table>

Home directories

The place you start when you log in. Home directory should be used for init files, small config files, etc. It is however not suitable for storing calculation data. Home directories are backed up daily. You usually want to use scratch instead.

scratch and work: Lustre

Scratch is the big, high-performance, 2PB Triton storage. It is the primary place for calculations, data analyzes etc. It is not backed up but is reliable against hardware failures (RAID6, redundant servers), but not safe against human error. It is shared on all nodes, and has very fast access. It is divided into two parts, scratch (by groups) and work (per-user). In general, always change to $WRKDIR or a group scratch directory when you first log in and start doing work. (note: home and work may be deleted six months after your account expires: use a group-based space instead).

Lustre separates metadata and contents onto separate object and metadata servers. This allows fast access to large files, but induces a larger overhead than normal filesystems. See our small files page for more information.

See Storage: Lustre (scratch)
Local disks

Local disks are on each node separately. It is used for the fastest I/Os with single-node jobs and is cleaned up after job is finished. Since 2019, things have gotten a bit more complicated given that our newest (skl) nodes don’t have local disks. If you want to ensure you have local storage, submit your job with --gres=spindle.

See the Compute node local drives page for further details and script examples.

ramfs - fast and highly temporary storage

On login nodes only, $XDG_RUNTIME_DIR is a ramfs, which means that it looks like files but is stored only in memory. Because of this, it is extremely fast, but has no persistence whatsoever. Use it if you have to make small temporary files that don’t need to last long. Note that this is no different than just holding the data in memory, if you can hold in memory that’s better.

Other Aalto data storage locations

Aalto has other non-Triton data storage locations available. See Data storage and Data: outline, requesting space, requesting access for more info.

Quotas

All directories under /scratch (as well as /home) have quotas. Two quotas are set per-filesystem: disk space and file number.

Disk quota and current usage are printed with the command quota. ‘space’ is for the disk space and ‘files’ for the total number of files limit. There is a separate quota for groups on which the user is a member.

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>space</th>
<th>quota</th>
<th>limit</th>
<th>grace</th>
<th>files</th>
<th>quota</th>
<th>limit</th>
<th>grace</th>
</tr>
</thead>
<tbody>
<tr>
<td>/home</td>
<td>484M</td>
<td>977M</td>
<td>1075M</td>
<td>10264</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/scratch</td>
<td>3237G</td>
<td>200G</td>
<td>210G</td>
<td>-</td>
<td>158M</td>
<td>1M</td>
<td>1M</td>
<td>-</td>
</tr>
</tbody>
</table>

Group quotas

<table>
<thead>
<tr>
<th>Filesystem</th>
<th>group</th>
<th>space</th>
<th>quota</th>
<th>limit</th>
<th>grace</th>
<th>files</th>
<th>quota</th>
</tr>
</thead>
<tbody>
<tr>
<td>/scratch</td>
<td>domain</td>
<td>132G</td>
<td>10M</td>
<td>10M</td>
<td>-</td>
<td>310M</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>-5000</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/scratch</td>
<td>some-group</td>
<td>534G</td>
<td>524G</td>
<td>524G</td>
<td>-</td>
<td>7534</td>
<td>1000M</td>
</tr>
<tr>
<td></td>
<td>-1000M</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/scratch</td>
<td>other-group</td>
<td>16T</td>
<td>20T</td>
<td>20T</td>
<td>-</td>
<td>1088M</td>
<td>5M</td>
</tr>
<tr>
<td></td>
<td>-5M</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If you get a quota error, see the quotas page for a solution.
Data availability throughout Aalto

Data is the basis of almost everything we do, and accessing it seamlessly throughout Aalto is a great benefit. Various other Aalto systems have the data available. However, this varies per department: each department can manage its data as it likes. So, we can’t make general promises about what is available where.

Linux shell server mounts require a valid Kerberos ticket (usually generated when you log in). On long sessions these might expire, and you have to renew them with `kinit` to keep going. If you get a permission denied, try `kinit`.

vdi.aalto.fi

vdi.aalto.fi has scratch mounted at /m/triton/scratch/. Your work folder can be access at /m/triton/scratch/work/USERNAME. For SCI departments the standard paths you have on your workstations are also working /m/{cs,nbe}/{scratch,work}/.

Shell servers

Departments have various shell servers, see below. There isn’t a generally available shell server anymore.

NBE

On workstations, work directories are available at /m/nbe/work and group scratch directories at /m/nbe/scratch/$project/, including the shell server amor.org.aalto.fi.

PHYS

Directories available on demand through SSHFS. See the Data transferring page at PHYS wiki.

CS

On workstations, work directories are available at /m/cs/work/, and group scratch directories at /m/cs/scratch/$project/. The department shell server is magi.cs.aalto.fi and has these available.

Remote access

There are many ways to access Triton data remotely. These days, we recommending figuring out how to `mount` the data remotely, so that it appears as local data but is accessed over the network. This saves copying data back and forth and is better for data security, but is slower and less reliable than local data.
Remote mounting using SMB

By far, remote mounting of files is the easiest method to transfer files. If you are not on the Aalto networks (wired, eduroam, or aalto with Aalto-managed laptop), connect to the Aalto VPN first. Note that this is automatically done on some department workstations (see below) - if not, request it!

The scratch filesystem can be remote mounted using SMB inside secure Aalto networks at the URLs

- scratch: smb://data.triton.aalto.fi/scratch/.
- work: smb://data.triton.aalto.fi/work/$username/.

On different operating systems:

- Linux (Ubuntu for example): File manager (Nautilus) → File → Connect to server. Use the smb:// URLs above.

- Windows: In the file manager, go to Computer (in menu bar on top, at least in Windows 10) → Map Network Drive) and “Map Network Drive”. In Windows 10 → “This PC” → right click → “Add Network Location”. (Note that this is different from right-click “Add network location” which just makes a folder link and has had some problems in the past.) Use the URLs above but replace smb:// with \\ and / with \. For example, \\data.triton.aalto.fi\scratch\.

- Mac: Finder → Go → Connect to Server. Use the smb:// URLs above.

- From Aalto managed computers, you can use lgw01.triton.aalto.fi instead of data.triton.aalto.fi and it might auto-login.

Depending on your OS, you may need to use either your username directly or AALTO\username.

Warning: In the future, you will only be able to do this from Aalto managed computers. This remote mounting will really help your work, so we recommend you to request an Aalto managed computer (citing this section) to make your work as smooth as possible (or use vdi.aalto.fi, see below).

Via vdi.aalto.fi

This is more like remote working, not remote access, but is useful here anyway. It doesn’t provide easy ways to upload/download files.

vdi.aalto.fi (described under Connecting to Triton) has Triton directories mounted (see below).

Note: work directories can be accessed via /m/triton/scratch/work/.

Remote mounting using sshfs

sshfs is a neat program that lets you mount remote filesystems via ssh only. It is well-supported in Linux, and somewhat on other operating systems. Its true advantage is that you can mount any remote ssh server - it doesn’t have to be set up specially for SMB or any other type of mounting. On Ubuntu, you can mount by “File → Connect to server” and using sftp://triton.aalto.fi/scratch/work/USERNAME. This also works from any shell server with data (see previous section).

The below uses command line programs to do the same, and makes the triton_work on your local computer access all files in /scratch/work/USERNAME. This can be done with other folders:

```bash
mkdir triton_work
sshfs USERNAME@triton.aalto.fi:/scratch/work/USERNAME triton_work
```
Note that ssh binds together many ways of accessing Triton, with a similar syntax and options. ssh is a very important program and binds together all types of remote access, and learning to use it well will help you for a long time. Learn more about ssh on the ssh page.

For Aalto Linux workstation users: it is recommended that you mount /scratch/ under the local disk /l/. You should be able to create the subfolder folder under /l/ and point sshfs to that subfolder as in the example here above.

**Using sftp**

The SFTP protocol uses ssh to transfer files. On Linux and Mac, the sftp command line program are the must fundamental way to do this, and are available everywhere.

A more user-friendly way of doing this (with a nice GUI) is the Filezilla program. Make sure you are using Aalto VPN, then you can put triton.aalto.fi as SFTP server with port 22.

Below is an example of the “raw” SFTP usage:

```
# Copying from HOME to local PC
user@pc123 $ sftp user12@triton.aalto.fi:filename
Connected to triton.aalto.fi.
Fetching /home/user12/filename to filename
# copying to HOME
user@pc123 $ sftp -b - user12@triton <<< 'put testCluster.m'
sftp> put foo
# copying to WRKDIR
user@pc123 $ sftp -b - user12@triton:/scratch/work/USERNAME/ <<< 'put testCluster.m'
...
```

With all modern OS it is also possible to just open your OS file manager (e.g. Nautilus on Linux) and just put as address in the bar:

```
sftp://triton.aalto.fi
```

If you are connecting from remote and cannot use the VPN, you can connect instead to department machines like kosh.aalto.fi, amor.org.aalto.fi (for NBE). The port is 22. *Note:* If you do not see your shared folder, you need to manually specify the full path (i.e. the folder is there, just not yet visible).

**Using rsync**

Rsync is similar to sftp, but is smarter at restarting files. Use rsync for large file transfers. rsync actually uses the ssh protocol so you can rsync from anywhere you can ssh from. rsync is installed by default on Linux and Mac terminals. On Windows machines we recommend using GIT-bash.

While there are better places on the internet to read about rsync, it is good to try it out to synchronise a local folder on your triton’s scratch. Sometimes the issue with copying files is related to group permissions. This command takes care of permissions and makes sure that all your local files are identical (= same MD5 fingerprint) to your remote files:

```
rsync -avzc -e "ssh" --chmod=g+s,g+rw --group=GROUPNAME PATHTOLOCALFOLDER
    --USERNAME@triton.aalto.fi:/scratch/DEPT/PROJECTNAME/REMOTEFOLDER/
```

Replace the bits in CAPS with your own case. Briefly, -a tries to preserve all attributes of the file, -v increases verbosity to see what rsync is doing, -z uses compression, -c skips files that have identical MD5 checksum, -e specifies to use ssh (not necessary but needed for the commands coming after), --chmod sets the group permissions to shared (as common practice on scratch project folders), and --group sets the groupname to the group you belong to (note that GROUPNAME == PROJECTNAME on our scratch filesystem).
If you want to just check that your local files are different from the remote ones, you can run `rsync` in “dry run” so that you only see what the command would do, without actually doing anything:

```
rsync --dry-run -avzc ...
```

Sometimes you want to copy only certain files. E.g. go through all folders, consider only files ending with `.py`:

```
rsync -avzc --include '*/' --include '*.py' --exclude '*' ...
```

Sometimes you want to copy only files under a certain size (e.g. 100MB):

```
rsync -avzc --max-size=100m ...
```

Rsync does NOT delete files by default, i.e. if you delete a file from the local folder, the remote file will not be deleted automatically, unless you specify the `--delete` option.

Please note that when working with files containing code or simple text, git is a better option to synchronise your local folder with your remote one, because not only it will keep the two folders in sync, but you will also gain version controlling so that you can revert to previous version of your code, or txt/csv files.

**Exercises**

**Data storage locations:**

1. *(Optional)* Look at the list of data storage locations above. Also look at the Data storage. Which do you think are suitable for your work? Do you need to share with others? Ask your group what they use and if you can use that, too.

**Note:** Many of the following exercises don’t work out of the box on other sites (depends on local files).

**Remote access:**

2. Mount your work directory by SMB - and alternatively sftp or sshfs - and transfer a file to Triton. Note that you must be connected to the Aalto VPN (from outside campus), or on eduroam, the aalto with Aalto laptop (from campus).

3. *(Advanced)* If you have a Linux on Mac computer, study the `rsync` manual page and try to transfer a file.

**About filesystem performance:**

`strace` is a command which tracks system calls, basically the number of times the operating system has to do something. It can be used as a rudimentary way to see how much I/O load there is.

4. Use `strace -c` to compare the number of system calls in `ls`, `ls -l`, `ls --no-color`, and `ls --color`. You can use the directory `/scratch/scip/lustre_2017/many-files/` as a place with many files in it. How many system calls per file were there for each option?

5. Using `strace -c`, compare the times of `find` and `lfs find` on the directory mentioned above. Why is it different?

6. *(Advanced, requires slurm knowledge from future tutorials)* You will find some sample files in `/scratch/scip/hpc-examples/io`. Create a temporary directory and...
   a) Run `create_iodata.sh` to make some data files in `data/`
   b) Compare the IO operations of `find` and `lfs find` on this directory.
   c) use the `iotest.sh` script to do some basic analysis. How long does it take? Submit it as a slurm batch job.
d) Modify the iotest.sh script to copy the `data/` directory to local storage, do the operations, then remove the data. Compare to previous strategy.

e) Use `tar` to compress the data while it is on lustre. Unpack this tar archive to local storage, do the operations, then remove. Compare to previous strategies.

Misc:

7. What do all of the following have in common?

   a) A job is submitted but fails with no output or messages.
   b) I can’t start a Jupyter server on jupyter.triton.
   c) Some files are randomly empty. Or the file had content, I tried to save it again, and now it’s empty!
   d) I can’t log in.
   e) I can log in with ssh, but `ssh -X` doesn’t work for graphical programs.
   f) I get an error message about corruption, such as `InvalidArchiveError("Error with archive ... You probably need to delete and re-download or re-create this file.

   g) I can’t install my own Python/R/etc libraries.

What’s next?

See also:

- Storage: Lustre (scratch)
- Storage: local drives
- Quotas
- Small files
- If you are doing heavy I/O: Storage

The next tutorial is about interactive jobs.

Interactive jobs

Video

Watch this in the Winter Kickstart 2021 course

Introduction to Slurm

Triton is a large system that combines many different individual computer nodes. Hundreds of people are using Triton simultaneously. Thus resources (CPU time, memory, etc.) need to be shared among everyone.

This resource sharing is done by a software called a job scheduler or workload manager. Triton’s workload manager is `slurm`. Triton users submit jobs which are then scheduled and allocated resources by the workload manager.

There are two ways you can submit your jobs to slurm queue system: either interactively using `srun` or by submitting a script using `sbatch`.
This tutorial walks you through running your jobs interactively. And in the next tutorial we will go through the more common and advanced way of submitting jobs, batch scripts.

An analogy: the HPC Diner

You’re eating out at the HPC Diner. What happens when you arrive?

- A host greets you and takes your party size and estimated dining time.
- You are given a number and asked to wait a bit.
- The host looks at who is currently waiting.
- If you are two people, you might squeeze in soon.
- If you are a lot of people, the host will try to slowly free up enough tables to join to eat together.
- If you are a really large party, you might need an advance reservation (or have to wait a really long time).
- They want everyone to get a fair share of their food. Thus, people that have visited more often are asked to wait slightly longer for their table, as a balancing mechanic.

Thanks to HPC Carpentry / Sabry Razick for the idea.

Your first interactive job

Let’s say you want to run the following command:

```
python3 -c 'import os; print("hi from", os.uname().nodename)'
```

You can submit this program to Triton using `srun`. All input/output still goes to your terminal (but note that graphical applications don’t work this way - see below):

```
$ srun --mem=100M --time=1:00:00 python3 -c 'import os; print("hi from", os.uname().nodename)'
```

srun: job 52204499 queued and waiting for resources

Here, we are asking for 100 Megabytes of memory (`--mem=100M`) for a duration of an hour (`--time=1:00:00`). While your job - with `jobid` 52204499 - is waiting to be allocated resources, your shell effectively become non-interactive.

You can open a new shell on triton and run the command `slurm q` to see all the jobs you have submitted to the queue:

```
$ slurm q
```

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>TIME</th>
<th>START_TIME</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>52204499</td>
<td>short-ivb</td>
<td>python3</td>
<td>0:00</td>
<td>N/A</td>
<td>PENDING</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

You can see information such as the state, which partition the requested node reside in, etc.

**Note:** The fact that we had to open another shell can be impractical if you need to run other jobs or just simply use the current shell. Additionally, if your shell quits while waiting (your internet may disconnect), the process cancels and you have to run the `srun` command again.
Once resources are allocated to your job, you see the name of the machine in the Triton cluster your program ran on, output to your terminal:

```
srun: job 52204499 has been allocated resources
hi from ivb17.int.triton.aalto.fi
```

**Note:** Interactive jobs are useful for debugging purposes, to test your setup and configurations before you put your tasks in a batch script for later execution. Or if you need graphical applications - such as Matlab, RStudio, etc. Additionally, if your task is small and not worth writing a batch script for, interactive job is the way to go. Keep in mind that you shouldn’t open 20 shells to run 20 `srun` jobs at once. Please have a look at the next tutorial about serial jobs.

---

**Interactive shell**

What if you want an actual shell to do things interactively? Put more precisely, you want access to a node in the cluster through an interactive bash shell that has all of the requested resources available. For this, you just need `srun`’s `--pty` option coupled with the shell you want:

```
srun -p interactive --time=2:00:00 --mem=600M --pty bash
```

The command prompt will appear when the job starts. And you will have a bash shell running on one of the computation nodes with at least 600 Megabytes of memory, for a duration of 2 hours, where you can run your programs in.

**Warning:** Remember to exit the shell when you are done! The shell will be running if you don’t and it will count towards your usage. This effectively means your priority will degrade in the future.

The option `--p interactive` requests a node in the interactive partition which is dedicated to interactive usage (more on this later). A partition is a group of nodes you can run on, with set limits.

**Note:** you can use `sinfo` to see information such as the available partitions, number of nodes in each, their time limits, etc.

---

**Interactive shell with graphics**

`sinteractive` is very similar to `srun`, but more clever and thus allows you to do X forwarding. It starts a screen session on the node, then sshes to there and connects to the screen. You can also ssh to this node again and connect to the process again.

```
sinteractive --time=1:00:00 --mem=1000M
```

**Warning:** Just like with `srun --pty bash`, remember to exit the shell. Since there is a separate screen session running, just closing the terminal isn’t enough. Exit all shells in the screen session on the node (C-d or exit), or cancel the process.
Note: If you are off-campus, you might want to use https://vdi.aalto.fi as a virtual desktop to connect to Triton to run graphical programs. Otherwise, programs may run very slowly.

Monitoring your usage

When your jobs enter the queue, you need to be able to get information on how much time, memory, etc. your jobs are using in order to know what requirements to ask for.

The command `slurm history` gives you information such as the actual memory used by your recent jobs, total CPU time, etc. You will learn more about these commands later on.

As shown in a previous example, the command `slurm queue` will tell you the currently running processes, which is a good way to make sure you have stopped everything.

Note: Generally, estimating the amount of time or memory you need comes down to monitoring you slurm history and utilizing command-line tools such as `time` on a few of your jobs and averaging. This is basically a trial and error process.

Setting resource parameters

Slurm comes with a multitude of parameters which you can specify to ensure you will be allocated enough memory, CPU cores, time, etc. You saw two of them in use in the above examples (`--mem` and `--time`) and you will learn more in the following tutorials.

Because you are sharing resource with other users, you should always estimate the amount of time, memory, etc. you need and then request them accordingly for efficiency reasons; the default memory and time limits are intentionally set low and may not be sufficient for your jobs to run/finish.

The general rule of thumb is to request the least possible, so that your stuff can run faster. That is because the less you request, the faster you are likely to be allocated resources. If you request something slightly less than a node size (note that we have different size nodes) or partition limit, you are more likely to fit into a spare spot.

For example, we have many nodes with 12 cores, and some with 20 or 24. If you request 24 cores, you have very limited options. However, you are more likely to be allocated a node if you request 10 cores. The same applies to memory: most common cutoffs are 48, 64, 128, 256GB. It’s best to use smaller values when submitting interactive jobs, and more for batch scripts.

See also:

This reference page covers the existing resource parameters and options you can use in both your interactive jobs and batch jobs which you will learn about in the next tutorial.
Exercises

The scripts you need for the following exercises can be found in this git repository: hpc-examples. You can clone the repository by running git clone https://github.com/AaltoSciComp/hpc-examples.git. This repository will be used for most of the tutorial exercises.

1. The program hpc-examples/slurm/memory-hog.py uses up a lot of memory to do nothing. Let's play with it. It's run as follows: python hpc-examples/slurm/memory-hog.py 50M, where the last argument is however much memory you want to eat. You can use --help to see the options of the program.
   a) Try running the program with 50M.
   b) Run the program with 50M and srun --mem=500M.
   c) Increase the amount of memory the Python process tries to use (not the amount of memory Slurm allocates). How much memory can you use before the job fails?
   d) Look at the job history using slurm history - can you see how much memory it actually used? - Note that Slurm only measures memory every 60 seconds or so. To make the program last longer, so that the memory used can be measured, give the --sleep option to the Python process, like this: python hpc-examples/slurm/memory-hog.py 50M --sleep=60 - keep it available.

2. The program hpc-examples/slurm/pi.py calculates pi using a simple stochastic algorithm. The program takes one positional argument: the number of trials.
   a) Run the program, timing it with time, a few times, increasing the number of trials, until it takes about 10 seconds: time python hpc-examples/slurm/pi.py 500, then 5000, then 50000, and so on.
   b) Add srun in front (srun python ...). Use the seff <jobid> command to see how much time the program took to run. (If you'd like to use the time command, you can run srun --mem=<m> --time=<t> time python hpc-examples/slurm/pi.py <iters>)
   c) Tell srun to use five CPUs (-c 5). Does it go any faster?
   d) Use the --threads=5 option to the Python program to tell it to also use five threads. ... python .../pi.py --threads=5
   e) Look at the job history using slurm history - can you see how much time each process used? What's the relation between TotalCPUTime and WallTime?

3. Check out some of these commands: sinfo, sinfo -N, squeue. Run slurm job <jobid> on some running job - does anything look interesting?

4. Run scontrol show node csl1 What is this? (csl1 is the name of a node on Triton - if you are not on Triton, look at the sinfo -N command and try one of those names).

What's next?

In the next tutorial on serial batch jobs, you will learn how to put the above-mentioned commands in a script, namely a batch script (a.k.a submission script) that allows for a multitude of jobs to run unattended.
Serial Jobs

Introduction to batch scripts

You learned, in the interactive jobs, how all Triton users must do their computation by submitting jobs to the Slurm batch system to ensure efficient resource sharing.

You additionally learned the interactive way to submit jobs, e.g. you could simply have an interactive Bash session on a compute node. This proves useful for tests and debugging. Slurm jobs, however, are normally batch jobs, meaning that they are run unattended and asynchronously, without human supervision.

To create a batch job, you need to create a job script and subsequently submit it to Slurm. A job script is simply a shell script, e.g. Bash, where you put your resource requests and job steps. You will see what these two components are in this tutorial. You have already seen how to do these interactively; and in this tutorial you will learn how to bundle them in your job scripts.

See also:
Please refer to the interactive jobs tutorial to learn the basics of Slurm.

Your first job script

A job script is simply a shell script (Bash). And so the first line in the script should be the shebang directive (#!) followed by the full path to the executable binary of the shell’s interpreter, which is Bash in our case. What then follow are the resource requests and the job steps.

Let’s take a look at the following script

```
#!/bin/bash
#SBATCH --time=00:05:00
#SBATCH --mem-per-cpu=100M
#SBATCH --output=hello.out
srun echo "Hello $USER! You are on node $HOSTNAME"
```

Let’s name it hello.sh (create a file using your editor of choice, e.g. nano; write the script above and save it)

The symbol # is a comment in a bash script, and Slurm understands #SBATCH as parameters, determining the resource requests. Here, we have requested a time limit of 5 minutes, along with 100 MB of RAM per CPU.

Resource requests are followed by job steps, which are the actual tasks to be done. Each srun within the a slurm script is a job step, and appears as a separate row in your history - which is useful for monitoring.

Having written the script, you need to submit the job to Slurm through the sbatch command:

```
$ sbatch hello.sh
Submitted batch job 52428672
```

Warning: You must use sbatch, not bash to submit the job since it is Slurm that understands the SBATCH directives, not Bash.
When the job enters the queue successfully, the response that the job has been submitted is printed in your terminal, along with the *jobid* assigned to the job.

You can check the status of your jobs using `slurm q`/`slurm queue` (or `squeue -u $USER`):

```
$ slurm q
+ JOBID PARTITION NAME     TIME     START_TIME STATE
- 52428672 debug  hello.sh  0:00      N/A    PENDING
```

Once the job is completed successfully, the state changes to *COMPLETED* and the output is then saved to `hello.out` in the current directory. You can also wildcards like `%u` for your username and `%j` for the jobid in the output file name. See the documentation of `sbatch` for a full list of available wildcards.

### Setting resource parameters

In both the above example and the tutorial on *interactive jobs*, you learned that resources are requested through job parameters such as `--mem`, `--time`, etc.

**See also:**

See *interactive jobs*, the *reference page* or the *details page* for more information and advanced usage.

Please keep in mind that these parameters are hard values. If, for example, you request 5 GB of memory and your job uses substantially more, Slurm will kill your job.

**Note:** Actually, there is a little bit of grace period in killing jobs (about an hour), and you can go over memory a little bit. But, if you go over the memory limit and the node runs out, you will be the first one to be killed! Don’t count on this.

We recommend you be as specific as possible when setting your resource parameters as they determine how fast your jobs will run. Therefore, please try to gain more understanding on how much resources your code needs to fine-tune your requested resources.

**Note:** In general, please do not submit too short jobs (under 5 minutes) unless you are debugging. For your bulk production, try to have each job take at least 30 minutes, if possible. The reason behind this is that there is a big amount of startup, accounting, and scheduling overhead.

### Monitoring your jobs

Once you submit your jobs, it goes into a queue. The two most useful commands to see the status of your jobs with are `slurm q`/`slurm queue` and `slurm h`/`slurm history` (or `squeue -u $USER` and `sacct -u $USER`). More information is in the *monitoring tutorial*. 

---

6.1. Triton cluster
Cancelling your jobs

You can cancel jobs with `scancel <jobid>`. To obtain job id, use the monitoring commands.

Partitions

A slurm partition is a set of computing nodes dedicated to a specific purpose. Examples include partitions assigned to debugging ("debug" partition), batch processing ("batch" partition), GPUs ("gpu" partition), etc.

Command `sinfo -s` lists a summary of the available partitions. For the sake of brevity, let’s see the first 4 partitions:

```
$ sinfo -s | head -n 5
PARTITION   AVAIL   TIMELIMIT NODES(A/I/O/T) NODELIST
interactive up  1-00:00:00  4/0/0/4 pe[4-7]  
jupyter-long up 10-00:00:00 4/0/0/4 pe[4-7]  
jupyter-short up 1-00:00:00 4/0/0/4 pe[4-7]  
grid        up  3-00:00:00 29/18/1/48 pe[9-48,74-81]  
```

Take a look at the manpage using `man sinfo` for more details.

Generally, you don’t need to specify the partition; Slurm will use any possible partition (though this is Aalto-specific, however other sites may have other requirements here). However, you can do so with `-p PARTITION_NAME`. This is mainly needed if you want to force interactive or debug partition (Slurm usually runs short jobs on the debug partition).

See also:

You can see the partitions in the quick reference.

Full reference

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sbatch</code></td>
<td>submit a job to queue (see standard options below)</td>
</tr>
<tr>
<td><code>srun</code></td>
<td>Within a running job script/environment: Run code using the allocated resources (see options below)</td>
</tr>
<tr>
<td><code>srun</code></td>
<td>On front end: submit to queue, wait until done, show output. (see options below)</td>
</tr>
<tr>
<td><code>sinteractive</code></td>
<td>Submit job, wait, provide shell on node for interactive playing (X forwarding works, default partition interactive). Exit shell when done. (see options below)</td>
</tr>
<tr>
<td><code>srun</code> --pty bash</td>
<td>(advanced) Another way to run interactive jobs, no X forwarding but simpler. Exit shell when done.</td>
</tr>
<tr>
<td><code>scancel &lt;jobid&gt;</code></td>
<td>Cancel a job in queue</td>
</tr>
<tr>
<td><code>salloc</code></td>
<td>(advanced) Allocate resources from frontend node. Use <code>srun</code> to run using those resources, exit to close shell when done. Read the description! (see options below)</td>
</tr>
<tr>
<td><code>scontrol</code></td>
<td>View/modify job and slurm configuration</td>
</tr>
<tr>
<td>Command</td>
<td>Option</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>sbatch</td>
<td>/etc -t, --time=hh:mm:ss</td>
</tr>
<tr>
<td></td>
<td>-t, --time=dd-hh</td>
</tr>
<tr>
<td></td>
<td>-p, --partition=partition</td>
</tr>
<tr>
<td></td>
<td>--mem-per-cpu=n</td>
</tr>
<tr>
<td></td>
<td>--mem=n</td>
</tr>
<tr>
<td></td>
<td>-c, --cpus-per-task=n</td>
</tr>
<tr>
<td></td>
<td>-N, --nodes=n-m</td>
</tr>
<tr>
<td></td>
<td>-n, --ntasks=n</td>
</tr>
<tr>
<td></td>
<td>-J, --job-name=name</td>
</tr>
<tr>
<td></td>
<td>-o output</td>
</tr>
<tr>
<td></td>
<td>-e error</td>
</tr>
<tr>
<td></td>
<td>--exclusive</td>
</tr>
<tr>
<td></td>
<td>--constraint=feature</td>
</tr>
<tr>
<td></td>
<td>--array=0-5,7,10-15</td>
</tr>
<tr>
<td></td>
<td>--gres=gpu</td>
</tr>
<tr>
<td></td>
<td>--gres=spindle</td>
</tr>
<tr>
<td></td>
<td>--mail-type=</td>
</tr>
<tr>
<td></td>
<td>--mail-user=only</td>
</tr>
<tr>
<td>srun</td>
<td>-N &lt;N_nodes&gt; hostname</td>
</tr>
</tbody>
</table>

See also:
There is a full description of running jobs on Triton and the reference page lists many useful commands.

Exercises
The scripts you need for the following exercises can be found in this git repository: hpc-examples. You can clone the repository by running `git clone https://github.com/AaltoSciComp/hpc-examples.git`. This repository will be used for most of the tutorial exercises.

1. Submit a batch job that just runs `hostname`.
   a. Set time to 1 hour and 15 minutes, memory to 500MB.
   b. Change the job’s name and output file.

6.1. Triton cluster
c. Check the output. Does the printed hostname match the one given by `slurm history/sacct -u $USER`?

2. Create a batch script which does nothing (or some pointless operation for a while), for example `sleep 300`. Check the queue to see when it starts running. Then, cancel the job. What output is produced?

3. Create a slurm script that runs the following program:

```bash
for i in $(seq 30); do
date
sleep 10
done
```

   a. Submit the job to the queue.

   b. Log out from Triton. Log back in and use `slurm queue/squeue -u $USER` to check the job status.

   c. Use `cat name_of_outputfile` to check at the output periodically.

   d. Cancel the job once you’re finished.

4. (Advanced) What happens if you submit a batch script with `bash` instead of `sbatch`? Does it appear to run? Does it use all the Slurm options?

5. (Advanced) Create a batch script that runs in another language using a different `#!/` line. Does it run? What are some of the advantages and problems here?

What’s next?

There are various tools one can use to do job monitoring.

Monitoring job progress and job efficiency

Introduction

When running jobs, one usually wants to do monitoring at various different stages:

- Firstly, when job is submitted, one wants to monitor the position of the job in the queue and expected starting time for the job.
- Secondly, when job is running, one wants to monitor the jobs state and how the simulations is performing.
- Thirdly, once the job has finished, one wants to monitor the job’s performance and resource usage.

There are various tools available for each of these steps.

See also:

Please ensure you have read Interactive jobs and Serial Jobs before you proceed with this tutorial.
Monitoring job queue state after it has been submitted

The command `slurm q`/`slurm queue` (or `squeue -u $USER`) can be used to monitor the status of your jobs in the queue. An example output is given below:

```
$ slurm q
JOBID PARTITION NAME TIME START_TIME STATE
       NODELIST(REASON)
60984785 interacti _interactive 0:29 2021-06-06T20:41 RUNNING pe6
60984796 batch-csl hostname 0:00 N/A PENDING

Here the output are as follows:

• **JOBID** shows the id number that Slurm has assigned for your job.

• **PARTITION** shows the partition(s) that the job has been assigned to.

• **NAME** shows the name of the submission script / job step / command.

• **TIME** shows the amount of time of the job has run so far.

• **START_TIME** shows the start time of the job. If job isn’t currently running, Slurm will try to form an estimate on when the job will run.

• **STATE** shows the state of the job. Usually it is **RUNNING** or **PENDING**.

• **NODES** shows the names of the nodes where the program is running. If the job isn’t running, Slurm tries to give a reason why the job is not running.

When submitting a job one often wants to see if job starts successfully. This can be made easier by running `slurm w q`/`slurm watch queue` or `(watch -n 15 squeue -u $USER)`. This opens a watcher that prints the output of `slurm queue` every 15 seconds. This watcher can be closed with `<CTRL> + C`. Do remember to close the watcher when you’re not watching the output interactively.

To see all of the information that Slurm sees, one can use the command `scontrol show -d jobid <jobid>`. The `slurm queue` is a wrapper built around `squeue`-command. One can also use it directly to get more information on the job’s status. See `squeue`’s documentation for more information.

There are other commands to `slurm` that you can use to monitor the cluster status, job history etc.. A list of examples is given below:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slurm q;slurm qq</td>
<td>Status of your queued jobs (long/short)</td>
</tr>
<tr>
<td>slurm partitions</td>
<td>Overview of partitions (A/I/O/T=active,idle,other,total)</td>
</tr>
<tr>
<td>slurm cpus &lt;partition&gt;</td>
<td>list free CPUs in a partition</td>
</tr>
<tr>
<td>slurm history [1day,2hour,...]</td>
<td>Show status of recent jobs</td>
</tr>
<tr>
<td>seff &lt;jobid&gt;</td>
<td>Show percent of mem/CPU used in job</td>
</tr>
<tr>
<td>slurm j &lt;jobid&gt;</td>
<td>Job details (only while running)</td>
</tr>
<tr>
<td>slurm s;slurm ss &lt;partition&gt;</td>
<td>Show status of all jobs</td>
</tr>
<tr>
<td>sacct</td>
<td>Full history information (advanced, needs args)</td>
</tr>
</tbody>
</table>

Full slurm command help:

```
$ slurm

Show or watch job queue:
   slurm [watch] queue    show own jobs
```

(continues on next page)
slurm [watch] q  show user's jobs
slurm [watch] quick  show quick overview of own jobs
slurm [watch] shorter sort and compact entire queue by job size
slurm [watch] short sort and compact entire queue by priority
slurm [watch] full  show everything
slurm [w] [q|qq|ss|sf] shorthands for above!
slurm qos  show job service classes
slurm top [queue|all]  show summary of active users

Show detailed information about jobs:
slurm prio [all|short] show priority components
slurm j|job  show everything else
slurm steps  show memory usage of running srun job steps

Show usage and fair-share values from accounting database:
slurm h|history  show jobs finished since, e.g. "1day" (default)
slurm shares

Show nodes and resources in the cluster:
slurm p|partitions  all partitions
slurm n|nodes  all cluster nodes
slurm c|cpus  total cpu cores in use
slurm cpus  cores available to partition, allocated and free
slurm cpus jobs  cores/memory reserved by running jobs
slurm cpus queue  cores/memory required by pending jobs
slurm features  List features and GRES

Examples:
slurm q
slurm watch shorter
slurm cpus batch
slurm history 3hours

Other advanced commands (many require lots of parameters to be useful):

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>squeue</td>
<td>Full info on queues</td>
</tr>
<tr>
<td>sinfo</td>
<td>Advanced info on partitions</td>
</tr>
<tr>
<td>slurm nodes</td>
<td>List all nodes</td>
</tr>
</tbody>
</table>

Monitoring job while it is running

As the most common way of using HPC resources is to run non-interactive jobs, it is usually a good idea to make certain that the program that will be run will produce some output that can be used to monitor the jobs’ progress.

Typical way of monitoring the progress is to add print-statements that produce output to the standard output. This output is then redirected to the Slurm output file where it can be read by the user.

It is important to differentiate between different types of output:

- **Monitoring output** is usually print statements and it describes what the program is doing (e.g. “Loading data”, “Running iteration 31”), what is the state of the simulation (e.g. “Total energy is 4.232 MeV”, “Loss is 0.432”) and to get timing information (e.g. “Iteration 31 took 182s”). This output can then be used to see if the program works, if the simulation converges and to determine how long does it take to do different calculations.

- **Debugging output** is similar to monitoring output, but it is usually more verbose and writes the internal state of the program (e.g. values of variables). This is usually required during development stage of a program, but once the program works and longer simulations are needed, printing debugging output is not recommended.
• **Checkpoint output** can be used to resume the current state of the simulation in the case of unexpected situations such as bugs, network problems or hardware failures. These should be in binary data as this keeps the accuracy of the floating point numbers intact. In big simulations checkpoints can be large, so the frequency of taking checkpoints should not be too high. In iterative processes e.g. Markov chain, taking checkpoints can be very quick and can be done more frequently. In smaller applications it is usually good to take checkpoints if the program starts a different phase of the simulation (e.g. plotting after simulation). This minimizes loss of simulation time due to programming bugs.

• **Simulation output** is something that the program outputs when the simulation is done. When doing long simulations it is important to consider what output parameters do you want to output. One should include all parameters that might be needed so that the simulations do not need to be run again. When doing time series output this is even more important as e.g. averages, statistical moments cannot necessarily be recalculated after the simulation has ended. It is usually good idea to save a checkpoint at the end as well.

When creating monitoring output it is usually best to write it in a human-readable format and human-readable quantities. This makes it easy to see the state of the program.

### Checking job history after it has finished

The command `slurm h`/`slurm history` can be used to check the history of your jobs. Example output is given below:

```
$ slurm h

<table>
<thead>
<tr>
<th>JobID</th>
<th>JobName</th>
<th>Start</th>
<th>ReqMem</th>
<th>MaxRSS</th>
<th>TotalCPUTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>60984785</td>
<td><em>interactive</em></td>
<td>06-06 20:41:31</td>
<td>500Mc</td>
<td></td>
<td>00:01.739</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60984796</td>
<td>hostname</td>
<td>06-06 20:49:36</td>
<td>500Mc</td>
<td></td>
<td>00:00.016</td>
</tr>
</tbody>
</table>

Here the output are as follows:

- **JobID** shows the id number that Slurm has assigned for your job.
- **JobName** shows the name of the submission script / job step / command.
- **Start** shows the start time of the job.
- **ReqMem** shows the amount of memory requested by the job. The format is an amount in megabytes or gigabytes followed by `c` or `n` for memory per core or memory per node respectively.
- **MaxRSS** shows the maximum memory usage of the job as calculated by Slurm. This is measured in set intervals.
- **TotalCPUTime** shows the total CPU time used by the job. It shows the amount of seconds the CPUs were at full utilization. For single CPU jobs, this should be close to the **WallTime**. For jobs that use multiple CPUs, this should be close to the number of CPUs reserved times **WallTime**.
- **WallTime** shows the runtime of the job in seconds.
- **Tasks** shows the number of MPI tasks reserved for the job.
- **CPU** shows the number of CPUs reserved for the job.
- **Ns** shows the number of nodes reserved for the job.

6.1. Triton cluster 161
Aalto scientific computing guide

• Exit State shows the exit code of the command. Successful run of the program should return 0 as the exit code.
• Nodes shows the names of the nodes where the program ran.

The slurm history-command is a wrapper built around sacct-command. One can also use it directly to get more information on the job’s status. See sacct’s documentation for more information.

For example, command sacct --format=jobid,elapsed,ncpus,ntasks,state,MaxRss --jobs=<jobid> which will show information as indicated in the --format option (jobid, elapsed time, number of reserved CPUs, etc.). You can specify any field of interest to be shown using --format.

Monitoring job’s CPU and RAM usage efficiency after it has finished

You can use seff <jobid> to see what percent of available CPUs and RAM was utilized. Example output is given below:

```
$ seff 60985042
Job ID: 60985042
Cluster: triton
User/Group: tuomiss1/tuomiss1
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 2
CPU Utilized: 00:00:29
CPU Efficiency: 90.62% of 00:00:32 core-walltime
Job Wall-clock time: 00:00:16
Memory Utilized: 1.59 MB
Memory Efficiency: 0.08% of 2.00 GB
```

If your processor usage is far below 100%, your code may not be working correctly. If your memory usage is far below 100% or above 100%, you might have a problem with your RAM requirements. You should set the RAM limit to be a bit above the RAM that you have utilized.

You can also monitor individual job steps by calling seff with the syntax seff <jobid>.<job step>.

**Important:** When making job reservations it is important to distinguish between requirements for the whole job (such as --mem) and requirements for each individual task/cpu (such as --mem-per-cpu). E.g. requesting --mem-per-cpu=2G with --ntasks=2 and --cpus-per-task=4 will create a total memory reservation of (2 tasks)*(4 cpus / task)*(2GB / cpu)=16GB.

Monitoring job’s GPU utilization

When running a GPU job, you should check that the GPU is being fully utilized.

When your job has started, you can ssh to the node and run nvidia-smi. You can find your process by e.g. using htop and inspect the GPU-Util column. It should be close to 100%.

Once the job has finished, you can use slurm history to obtain the jobID and run:

```
$ sacct -j <jobID> -o comment -p
```

This also shows the GPU utilization.
Note: There are factors to be considered regarding efficient use of GPUs. For instance, is your code itself efficient enough? Are you using the framework pipelines in the intended fashion? Is it only using GPU for a small portion of the entire task? Amdahl’s law of parallelization speedup is relevant here.

If the GPU utilization of your job is low, you should check whether its CPU utilization is close to 100% with seff <jobid>. This can indicate that the CPUs are trying to keep the GPU occupied with calculations, but the lack of CPU performance will cause a bottleneck on the GPU utilization.

Please keep in mind that when using a GPU, you need to also request enough CPUs to supply the data to the process. So, you can increase the number of CPUs you request so that enough data is provided for the GPU. However, you shouldn’t request too many: There wouldn’t be enough CPUs for everyone to use the GPUs, and they would go to waste (all of our nodes have 4-6 CPUs for each GPU).

Exercises

The scripts you need for the following exercises can be found in this git repository: hpc-examples. You can clone the repository by running git clone https://github.com/AaltoSciComp/hpc-examples.git. This repository will be used for most of the tutorial exercises.

1. In folder slurm/pi.py there is a pi estimation algorithm that uses Monte Carlo methods to get an estimate of its value. You can call the script with python pi.py <n>, where <n> is the number of iterations to be done by the algorithm.
   a. Create a slurm script that runs the algorithm with 100000000 (10^8) iterations. Submit it to the queue and use slurm queue, slurm history and seff to monitor the job’s performance.
   b. Add multiple job steps (separate srun lines), each of which runs the algorithm pi.py with increasing number of iterations (from range 100 - 10000000 (10^7)). How does this appear in slurm history?
   c. Use seff to check performance of individual job steps. Can you explain why the CPU utilization numbers change between steps?

2. The script pi.py has been written so that it can be run using multiple threads. Run the script with multiple threads and 10^8 iterations with:
   
   srun --cpus-per-task=2 python pi.py --threads=2 100000000

   After you have run the script, do the following:
   a. Use slurm history to check the TotalCPUtil and WallTime. Compare them to the timings for the single CPU run with 10^8 iterations.
   b. Use seff to check CPU performance of the job.

What’s next?

Running multiple instances of a sbatch script is easier with array jobs.
Array jobs

Video

Watch this in the Winter Kickstart 2021 course

More often than not, scientific problems involve running a single program again and again with different datasets or parameters.

When there is no dependency or communication among the individual program runs, these individual runs can be run in parallel on separate Slurm jobs. This kind of parallelism is called **embarrassingly parallel**.

Slurm has a structure called **job array**, which enables users to easily submit and run several instances of the same Slurm script independently in the queue.

Introduction

Array jobs allow you to parallelize your computations. They are used when you need to run the same job many times with only slight changes among the jobs. For example, you need to run 1000 jobs each with a different seed value for the random number generator. Or perhaps you need to apply the same computation to a collection of data sets. These can be done by submitting a single array job.

Slurm job array is a collection of jobs that are to be executed with identical parameters. This means that there is one single batch script that is to be run as many times as indicated by the `--array` directive, e.g.:

```
#SBATCH --array=0-4
```

creates an array of 5 jobs (tasks) with index values 0, 1, 2, 3, 4.

The array tasks are copies of the “master” batch script that are automatically submitted to Slurm. Slurm provides a unique environment variable `SLURM_ARRAY_TASK_ID` to each task which could be used for handling input/output files to each task.

**Note:** You can alternatively pass the `--array` option as a command-line argument to `sbatch`.

**Important:** When running array job you’re basically running identical copies of a single job. Thus it is increasingly important to know how your code behaves with respect to the file system:

- Does it use libraries/environment stored in the work directory?
- How much input data does it need?
- How much output data does the job create?

For example, running an array job with hundreds of workers that uses a Python environment stored in the work disk can inadvertently cause a lot of filesystem load as there will be hundreds of thousands of file calls.

If you’re unsure how your job will behave, ask admins for help.
Your first array job

Let’s see a job array in action. Let’s create a file called `array_example.sh` and write it as follows.

```bash
#!/bin/bash
#SBATCH --time=00:15:00
#SBATCH --mem=200M
#SBATCH --output=array_example_%A_%a.out
#SBATCH --array=0-15

# You may put the commands below:
# Job step
srun echo "I am array task number" $SLURM_ARRAY_TASK_ID
```

Submitting the job script to Slurm with `sbatch array_example.sh`, you will get the message:

```
$ Submitted batch job 60997836
```

The job id in the message is that of the “master” job. This is common for all of the jobs in the array. In addition, each individual job is given an array task id.

As now we’re submitting multiple jobs simultaneously, each job needs an individual output file or the outputs will overwrite each other. By default, Slurm will write the outputs to files named `slurm-$(SLURM_ARRAY_JOB_ID)_$(SLURM_ARRAY_TASK_ID).out`. This can be overwritten using the `--output=<filename>` parameter, when you can use wildcard `%A` for the job id and `%a` for the array task id.

Once the jobs are completed, the output files will be created in your work directory, with the help `%u` to determine your user name:

```
$ ls
array_example_60997836_0.out array_example_60997836_12.out array_example_60997836_15.out array_example_60997836_3.out array_example_60997836_6.out array_example_60997836_9.out
array_example_60997836_10.out array_example_60997836_13.out array_example_60997836_1.out array_example_60997836_4.out array_example_60997836_7.out array_example.sh
array_example_60997836_11.out array_example_60997836_14.out array_example_60997836_2.out array_example_60997836_5.out array_example_60997836_8.out
```

You can `cat` one of the files to see the output of each task:

```
$ cat array_example_60997836_11.out
I am array task number 11
```

**Important:** The array indices do not need to be sequential. For example, if after running an array job you find out that tasks 2 and 5 failed, you can relaunch just those jobs with `--array=2,5`.

You can even simply pass the `--array` option as a command-line argument to `sbatch`.

---

6.1. Triton cluster
More examples

The following examples give you an idea on how to use job arrays for different use cases and how to utilize the
$SLURM_ARRAY_TASK_ID environment variable.

Reading input files

In many cases, you would like to process several data files. That is, pass different input files to your code to be
processed. This can be achieved by using $SLURM_ARRAY_TASK_ID environment variable.

In the example below, the array job gives the program different input files, based on the value of the
$SLURM_ARRAY_TASK_ID:

```
#!/bin/bash
#SBATCH --time=01:00:00
#SBATCH --mem=1G
#SBATCH --array=0-29

# Each array task runs the same program, but with a different input file.
# e.g. srun ./my_application -input input_data_${SLURM_ARRAY_TASK_ID}
```

Hardcoding arguments in the batch script

One way to pass arguments to your code is by hardcoding them in the batch script you want to submit to Slurm.

Assume you would like to run the pi estimation code for 5 different seed values, each for 2.5 million iterations. You
could assign a seed value to each task in your array job and save each output to a file. Having calculated all estimations,
you could take the average of all the pi values to arrive at a more accurate estimate. An example of such a batch script
pi_array_hardcoded.sh is as follows.

```
#!/bin/bash
#SBATCH --time=01:00:00
#SBATCH --mem=500M
#SBATCH --job-name=pi-array-hardcoded
#SBATCH --output=pi-array-hardcoded_%a.out
#SBATCH --array=0-4

case $SLURM_ARRAY_TASK_ID in
  0) SEED=123 ;;
  1) SEED=38  ;;
  2) SEED=22  ;;
  3) SEED=60  ;;
  4) SEED=432 ;;
esac

srun python slurm/pi.py 2500000 --seed=$SEED > pi_$SEED.json
```

Save the script and submit it to Slurm:

```
$ sbatch pi_array_hardcoded.sh
Submitted batch job 60997871
```

Once finished, 5 Slurm output files and 5 application output files will be created in your current directory each con-
taining the pi estimation; total number of iterations (sum of iteration per task); and total number of successes: 
Reading parameters from one file

Another way to pass arguments to your code via script is to save the arguments to a file and have your script read the arguments from it.

Drawing on the previous example, let’s assume you now want to run `pi.py` with different iterations. You can create a file, say `iterations.txt` and have all the values written to it, e.g.:

```bash
$ cat iterations.txt
100
1000
50000
1000000
```

You can modify the previous script to have it read the `iterations.txt` one line at a time and pass it on to `pi.py`. Here, `sed` is used to get each line. Alternatively you can use any other command-line utility, e.g. `awk`. Do not worry if you don’t know how `sed` works - Google search and `man sed` always help. Also note that the line numbers start at 1, not 0.

The script `pi_array_parameter.sh` looks like this:

```bash
#!/bin/bash
#SBATCH --time=01:00:00
#SBATCH --mem=500M
#SBATCH --job-name=pi-array-parameter
#SBATCH --output=pi-array-parameter_%a.out
#SBATCH --array=1-4

n=${SLURM_ARRAY_TASK_ID}
iteration=`sed -n "${n}p" iterations.txt` # Get n-th line (1-indexed) of the file
srun python slurm/pi.py ${iteration} > pi_iter_${iteration}.json
```

You can additionally do this procedure in a more complex way, e.g. read in multiple arguments from a csv file, etc.

(Advanced) Grouping runs together in bigger chunks

If you have lots of jobs that are short (a few minutes), using array jobs may induce too much overhead in scheduling and you will create huge number of output files. In these kinds of cases you might want to combine multiple program runs into a single array job.

Important: A good target time for the array jobs would be approximately 30 minutes, so please try to combine your tasks so that each job would at least take this long.

Easy workaround for this is to create a for-loop in your Slurm script. For example, if you want to run the pi script with 50 different seed values you could run them in chunks of 10 and run a total of 5 array jobs. This the amount of array jobs we need by a factor of 10!

This method demands more knowledge of shell scripting, but the end result is a fairly simple slurm script `pi_array_parameter.sh` that does what we need.

6.1. Triton cluster
#!/bin/bash
#SBATCH --time=01:00:00
#SBATCH --mem=500M
#SBATCH --job-name=pi-array-grouped
#SBATCH --output=pi-array-grouped_%a.out
#SBATCH --array=1-4

# Lets create a new folder for our output files
mkdir -p json_files

CHUNKSIZE=10
n=$SLURM_ARRAY_TASK_ID
indexes="`seq $((n*CHUNKSIZE)) $(((n + 1)*CHUNKSIZE - 1))`"

for i in $indexes
do
    srun python slurm/pi.py 1500000 --seed=$i > json_files/pi_${i}.json
done

Exercises

The scripts you need for the following exercises can be found in this git repository: hpc-examples. You can clone the repository by running git clone https://github.com/AaltoSciComp/hpc-examples.git. This repository will be used for most of the tutorial exercises.

1. Make an array job that runs slurm/memory-hog.py with five different values of memory (50M, 100M, 500M, 1000M, 5000M) using one of the techniques above - this is the memory that the memory-hog script requests, not the memory requested from Slurm. Request 250M of memory for the array job. See if some of the jobs fail.

2. Think about your typical work. How could you split your stuff into trivial pieces that can be run with array jobs?

3. (Advanced) Make a job array which runs every other index, e.g. the array can be indexed as 1, 3, 5... (sbatch manual page can be of help)

4. (Advanced) Create a job array that uses the slurm/pi.py to calculate a combination of different iterations and seed values and save them all to different files.

What's next?

See also:

For more information, you can see the CSC guide on array jobs

Please check the quick reference when needed.

If you need more detailed information about running on Triton, see the main page Running programs on Triton.

The next tutorial is about GPU computing.
GPU computing

Introduction

GPUs, short for graphical processing unit, are massively-parallel processors that are optimized to perform parallel operations. Computations that might take days to run on CPUs, take substantially less time on GPUs. This speed-up specially comes in handy when dealing with large amounts of data, e.g. in machine learning/deep learning tasks, which is why GPUs have become an indispensable tool in the research community.

The programs we normally write in common programming languages, e.g. C++ are executed by the CPU. We need to explicitly communicate with the GPU if we want GPU to execute the program. That is, upload the program and the input data to the GPU, and transfer the result from the GPU to the main memory. What enable this procedure are programming environments designed to communicate with GPUs in such a manner. An example of such an API is CUDA which is the native programming interface for NVIDIA GPUs.

On Triton, we have a large number of NVIDIA GPU cards from different generations and currently only support CUDA. Triton GPUs are not the typical desktop GPUs, but specialized research-grade server GPUs with large memory, high bandwidth and specialized instructions, that are constantly increasing in number. For scientific purposes, they generally outperform the best desktop GPUs.

See also:

Please ensure you have read Interactive jobs and Serial Jobs before you proceed with this tutorial.

GPU jobs

To request GPUs on Slurm, you should use the --gres option either in your batch script or as a command-line argument to your interactive job. Used with a SBATCH directive in a batch script, exactly one GPU is requested as follows. :

```
#SBATCH --gres=gpu:1
```

You can request as many GPUs as you’d like using #SBATCH --gres=gpu:<n> wherein n denotes the number of the requested GPUs.

Note: Most of the time, using more than one GPU isn’t worth it, unless you specially optimize, because communication takes too much time. It’s better to parallelize by splitting tasks into different jobs.

You can restrict yourself to a certain type of GPU card by using using the --constraint option. For example, to restrict to Kepler generation (K80s), use --constraint='kepler' or only Pascal or Volta generations with --constraint='pascal|volta' (Remember to use the quotes since | is the shell pipe)
Available machine learning frameworks

We support many common machine learning frameworks out of the box:

- **Tensorflow**: module load anaconda. See the Tensorflow page for info on older versions.
- **Keras**: module load anaconda
- **PyTorch**: module load anaconda

Please note that most of the pre-installed softwares have CUDA already present. Thus you **do not need to load CUDA** as a seperate module when loading these. See the application list or GPU computing reference for more details.

Compiling CUDA-based code

To compile CUDA-based code for GPUs, you need to load the relevant `cuda` module. You can see what versions of CUDA is available using `module spider`:

```
$ module spider cuda
```

When submitting a batch script, you need to load the `cuda` module, compile your code, and subsequently run the executable. An example of such a submission script is shown below wherein the output of the code is written to a file named `helloworld.out` in the current directory:

```
#!/bin/bash
#SBATCH --time=00:05:00
#SBATCH --job-name=helloworld
#SBATCH --mem-per-cpu=500M
#SBATCH --cpus-per-task=1
#SBATCH --gres=gpu:1
#SBATCH --output=helloworld.out

module load cuda
nvcc helloworld.cu -o helloworld
./helloworld
```

*Note:* If you ever get `libcuda.so.1: cannot open shared object file: No such file or directory`, this means you are attempting to use a CUDA program on a node without a GPU. This especially happens if you try to test GPU code on the login node, and happens (for example) even if you try to import the GPU tensorflow module in Python on the login node.

Examples

Simple Tensorflow/Keras model

Let’s run the MNIST example from Tensorflow’s tutorials:

```python
model = tf.keras.models.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28, 28)),  
    tf.keras.layers.Dense(512, activation=tf.nn.relu),  
    tf.keras.layers.Dropout(0.2),  
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)  
])
```
The full code for the example is in `tensorflow_mnist.py`. One can run this example with `srun`:

```
wget https://raw.githubusercontent.com/AaltoSciComp/scicomp-docs/master/triton/examples/tensorflow/tensorflow_mnist.py
module load anaconda
srun --time=00:15:00 --gres=gpu:1 python tensorflow_mnist.py
```

or with `sbatch` by submitting `tensorflow_mnist.sh`:

```
#!/bin/bash
#SBATCH --gres=gpu:1
#SBATCH --time=00:15:00
module load anaconda
python tensorflow_mnist.py
```

Do note that by default Keras downloads datasets to `$HOME/.keras/datasets`.

### Simple PyTorch model

Let’s run the MNIST example from PyTorch’s tutorials:

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

The full code for the example is in `tensorflow_mnist.py`. One can run this example with `srun`:

```
wget https://raw.githubusercontent.com/AaltoSciComp/scicomp-docs/master/triton/examples/pytorch/pytorch_mnist.py
module load anaconda
srun --time=00:15:00 --gres=gpu:1 python pytorch_mnist.py
```

or with `sbatch` by submitting `pytorch_mnist.sh`:

```
#!/bin/bash
#SBATCH --gres=gpu:1
#SBATCH --time=00:15:00
module load anaconda
python pytorch_mnist.py
```
The Python-script will download the MNIST dataset to data folder.

**Monitoring efficient use of GPUs**

When running a GPU job, you should check that the GPU is being fully utilized.

When your job has started, you can ssh to the node and run `nvidia-smi`. You can find your process by e.g. using `htop` and inspect the GPU-Util column. It should be close to 100%.

Once the job has finished, you can use `slurm history` to obtain the jobID and run:

$ sacct -j <jobID> -o comment -p

This also shows the GPU utilization.

**Note:** There are factors to be considered regarding efficient use of GPUs. For instance, is your code itself efficient enough? Are you using the framework pipelines in the intended fashion? Is it only using GPU for a small portion of the entire task? Amdahl’s law of parallelization speedup is relevant here.

If the GPU utilization of your job is low, you should check whether its CPU utilization is close to 100% with `seff <jobid>`. This can indicate that the CPUs are trying to keep the GPU occupied with calculations, but the lack of CPU performance will cause a bottleneck on the GPU utilization.

Please keep in mind that when using a GPU, you need to also request enough CPUs to supply the data to the process. So, you can increase the number of CPUs you request so that enough data is provided for the GPU. However, you shouldn’t request too many: There wouldn’t be enough CPUs for everyone to use the GPUs, and they would go to waste (all of our nodes have 4-6 CPUs for each GPU).

**Input/output**

Deep learning work is intrinsically very data-hungry. Remember what we said about storage and input/output being important before (Data storage)? This matter becomes very important when working with GPUs. In fact, faster memory bandwidth is the main improvement of our server-grade GPUs compared to desktop models.

If you are loading big amounts of data, you should package the data into a container format first; lots of small files are your worst enemy. Each framework has a way to do this efficiently in a whole pipeline.

**See also:**

Please refer to the small files page for more detailed information.

If your data consists of individual files that are not too big, it is a good idea to have the data stored in one file, which is then copied to nodes ramdisk `/dev/shm` or temporary disk `/tmp`.

If your data is too big to fit in the disk, we recommend that you contact us for efficient data handling models.
Available GPUs and architectures

<table>
<thead>
<tr>
<th>Card</th>
<th>total amount</th>
<th>nodes architecture</th>
<th>compute threads per GPU</th>
<th>memory per card</th>
<th>CUDA compute capability</th>
<th>Slurm feature name</th>
<th>Slurm gres name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tesla K80*</td>
<td>12</td>
<td>gpu[20-22]</td>
<td>Kepler</td>
<td>2x2496</td>
<td>2x12GB</td>
<td>3.7</td>
<td>kepler</td>
</tr>
<tr>
<td>Tesla P100</td>
<td>20</td>
<td>gpu[23-27]</td>
<td>Pascal</td>
<td>3854</td>
<td>16GB</td>
<td>6.0</td>
<td>pascal</td>
</tr>
<tr>
<td>Tesla V100</td>
<td>40</td>
<td>gpu[1-10]</td>
<td>Volta</td>
<td>5120</td>
<td>32GB</td>
<td>7.0</td>
<td>volta</td>
</tr>
<tr>
<td>Tesla V100</td>
<td>40</td>
<td>gpu[28-37]</td>
<td>Volta</td>
<td>5120</td>
<td>32GB</td>
<td>7.0</td>
<td>volta</td>
</tr>
<tr>
<td>Tesla V100</td>
<td>16</td>
<td>dgx[01-02]</td>
<td>Volta</td>
<td>5120</td>
<td>16GB</td>
<td>7.0</td>
<td>volta</td>
</tr>
</tbody>
</table>

Exercises

The scripts you need for the following exercises can be found in this git repository: hpc-examples. You can clone the repository by running git clone https://github.com/AaltoSciComp/hpc-examples.git. This repository will be used for most of the tutorial exercises.

1. Run nvidia-smi on a GPU node with srun. Use slurm history to check which GPU node you ended up on. Try setting a constraint to force a different GPU architecture.

2. Load cuda and gcc (version less than 9) modules and compile the gpu/pi.cu example using nvcc. Run it. Does it say zero? Try running it with a GPU and see what happens.

3. Run one of the samples given above. Try using sbatch as well.

4. (Advanced) The PyTorch example will try to load datasets from a folder called data in a local folder. Modify the Slurm script so that the script:
   a. Creates an unique folder in /dev/shm or $TMPDIR before running the Python code.
   b. Moves to this folder when job is running.
   c. Runs the PyTorch-example from this location. Verify that the datasets are stored in the local disk.
   HINT: Check out mktemp --help, command output substitutions-section from our Linux shell tutorial and the API page for Python’s os.environ.

What's next?

You can see the main article, GPU Computing, for more detailed information regarding GPU computing, including examples of different machine learning frameworks.

This guide assumes you are using pre-existing GPU programs. If you need to write your own, that’s a whole other story, and you can find some hints on the reference page.
Parallel computing is what HPC is really all about: processing things on more than one processor at once. By now, you should have read all of the previous tutorials.

Parallel programming models

Parallel programming is used to create programs that can execute instructions on multiple processors at a same time. Most of our users that run their programs in parallel utilize existing parallel execution features that are present in their programs and thus do not need to learn how to create parallel programs. But even when one is running programs in parallel, it is important to understand different models of parallel execution.

The two main models are:

- **Shared memory (or multithreaded/multiprocess) programs** run multiple independent workers on the same machine. As the name suggests, all of the computer’s memory has to be accessible to all of the processes. **Thus programs that utilize this model should request one node, one task and multiple CPUs.** Likewise, the maximum number of workers is usually the number of CPU cores available on the computational node. The code is easier to implement and the same code can still be run in a serial mode. Example applications that utilize this model: Matlab, R, Python multithreading/multiprocessing, OpenMP applications, BLAS libraries, FFTW libraries, typical multithreaded/multiprocess parallel desktop programs.

- **Message passing programming (e.g. MPI, message passing interface)** can run on multiple nodes interconnected with the network via passing data through MPI software libraries. Almost all large-scale scientific programs utilize MPI. MPI can scale to thousands of CPU cores, but depending on the case it can be harder to implement from the programmer’s point of view. **Programs that utilize this model should request single/multiple nodes with multiple tasks each. You should not request multiple CPUs per task.** Example applications that utilize this model: CP2K, GPAW, LAMMPS, OpenFoam.

Both models, MPI and shared memory, can be combined in one application, in this case we are talking about hybrid parallel programming model. **Programs that utilize this model can require both multiple tasks and multiple CPUs per task.**

Most historical scientific code is MPI, but these days more and more people are using shared memory models.

**Important:** Normal serial code can’t just be run in parallel without modifications. As a user it is your responsibility to understand what parallel model implementation your code has, if any.

When deciding whether using parallel programming is worth the effort, one should be mindful of Amdahl’s law and Gustafson’s law. All programs have some parts that can only be executed in serial and thus the theoretical speedup that one can get from using parallel programming depends on two factors:

1. How much of programs’ execution could be done in parallel?
2. What would be the speedup for that parallel part?

Thus if your program runs mainly in serial but has a small parallel part, running it in parallel might not be worth it. Sometimes, doing data parallelism with e.g. *array jobs* is much more fruitful approach.

Another important note regarding parallelism is that all the applications scale good up to some upper limit which depends on application implementation, size and type of problem you solve and some other factors. The best practice is to benchmark your code on different number of CPU cores before you start actual production runs.
If you want to run some program in parallel, you have to know something about it - is it shared memory or MPI? A program doesn’t magically get faster when you ask more processors if it’s not designed to.

**Shared memory: OpenMP/multithreaded/multiprocess**

**Difference between multithreaded and multiprocess**

Shared memory programs usually parallelize by using multiple threads or processes. Processes are individual program executions while threads are basically smaller program executions within a process. Processes can launch both subprocesses and threads. Slurm reservations for both methods behave similarly.

Depending on a program, you might have multiple processes (Matlab parallel pool, R parallel-library, Python multiprocessing) or have multiple threads (OpenMP threads of BLAS libraries that R/numpy use).

**Warning:** Some programs (e.g. R) can utilize both multithread and multiprocess parallelism. For example, R has parallel-library for running multiple processes, but BLAS libraries that R uses can utilize multiple threads. If you encounter bad performance when you use parallel processes try setting `OMP_NUM_THREADS=1` in your slurm script.

**Running multithreaded/multiprocess applications**

The basic slurm option that specifies how many CPUs your job requires is `--cpus-per-task=N` (or `--c N`). If your memory requirement scales with the number of cores, use `--mem-per-core=M`, if you require a fixed amount of memory (per node regardless of number of processors), use `--mem=M`. We recommend starting with `--mem=M` if you do not know how your problem scales.

**Important:** The number of threads/processes you launch should match the number of requested processors. If you create a lower number, you will not utilize all CPUs. If you launch a larger number, you will oversubscribe the CPUs and the code will run slower as different threads/processes will have to swap in/out of the CPUs.

**Warning:** Normally you should not use `--ntasks=N` when you want to run shared memory codes. The number of tasks is only relevant to MPI codes and by specifying it you might launch multiple copies of your program that all compete on the reserved CPUs.

Only hybrid parallelization codes should have both `--ntasks=N` and `--cpus-per-task=C` set to be greater than one.

**Running a typical OpenMP program**

OpenMP is a standard de facto for the multithreading implementations. There are many others, but this one is the most common, supported by all known compiler suits. For other implementations of shared memory parallelism, please consult your code docs.

Let’s consider `hello_omp-example` from HPC examples repository.

Simple code compiling:
wget https://raw.githubusercontent.com/AaltoSciComp/hpc-examples/master/openmp/hello_omp/hello_omp.c  
module load gcc/9.2.0  
gcc -fopenmp -O2 -g hello_omp.c -o hello_omp

Running an OpenMP code:

```
export OMP_PROC_BIND=TRUE  
module load gcc/9.2.0  
srun --cpus-per-task=4 --mem=500M --time=00:05:00 hello_omp
```

The slurm script will look similar:

```
#!/bin/bash -l
#SBATCH --time=00:05:00  
#SBATCH --mem=500M  
#SBATCH --cpus-per-task=4  
#SBATCH --output=hello_omp.out

module load gcc/9.2.0  
export OMP_PROC_BIND=true  
srun hello_omp
```

It is good to know that OpenMP is both an environment and set of libraries, but those libraries always come as part of the compiler. Thus during runtime you should load the same compiler that you used for compiling the code.

**Running Python with OpenMP parallelization**

Various Python packages such as Numpy, Scipy and pandas can utilize OpenMP to run on multiple CPUs. As an example, let’s run the python script `python_openmp.py` that calculates multiplicative inverse of five symmetric matrices of size 2000x2000.

```
nrounds = 5  
t_start = time()  
for i in range(nrounds):  
    a = np.random.random([2000,2000])  
    a = a + a.T  
    b = np.linalg.pinv(a)  
    t_delta = time() - t_start  
print('Seconds taken to invert %d symmetric 2000x2000 matrices: %f' % (nrounds, t_delta))
```

The full code for the example is in HPC examples-repository. One can run this example with srun:

```
wget https://raw.githubusercontent.com/AaltoSciComp/hpc-examples/master/python/python_openmp.py  
module load anaconda  
export OMP_PROC_BIND=true  
srun --cpus-per-task=2 --mem=2G --time=00:15:00 python python_openmp.py
```

or with sbatch by submitting `python_openmp.slrm`:

176 Chapter 6. Triton
#!/bin/bash -l
#SBATCH --time=00:10:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=2
#SBATCH --mem-per-cpu=1G
#SBATCH -o python_openmp.out

module load anaconda/2020-03-tf2
export OMP_PROC_BIND=true

echo 'Running on: ' $HOSTNAME

srun python python_openmp.py

**Important:** Python has a global interpreter lock (GIL), which forces some operations to be executed on only one thread and when these operations are occurring, other threads will be idle. These kinds of operations include reading files and doing print statements. Thus one should be extra careful with multithreaded code as it is easy to create seemingly parallel code that does not actually utilize multiple CPUs.

There are ways to minimize effects of GIL on your Python code and if you’re creating your own multithreaded code, we recommend that you take this into account.

**Message passing programs: MPI**

For compiling/running an MPI job one has to pick up one of the MPI library suites. There are various different MPI libraries that all implement the MPI standard. We recommend that you use either:

- OpenMPI (e.g. openmpi/3.1.4)
- Intel’s MPI (e.g. intel-parallel-studio/cluster.2020.0-intelmpi)

Some libraries/programs might have already existing requirement for a certain MPI version. If so, use that version or ask for administrators to create a version of the library with dependency on the MPI version you require.

**Warning:** Different versions of MPI are not compatible with each other. Each version of MPI will create code that will run correctly with only that version of MPI. Thus if you create code with a certain version, you will need to load the same version of the library when you are running the code.

Also, the MPI libraries are usually linked to slurm and network drivers. Thus, when slurm or driver versions are updated, some older versions of MPI might break. If you’re still using said versions, let us know. If you’re just starting a new project, it is recommended to use our recommended MPI libraries.

For basic use of MPI programs, you will need to use the `-n N`/`--ntasks=N`-option to specify the number of MPI workers.

---

6.1. Triton cluster
Running a typical MPI program

The following use hpc-examples from *the previous exercises*.

Loading module:

```bash
# GCC + OpenMPI
module load gcc/9.2.0  # GCC
module load openmpi/3.1.4  # OpenMPI

# Intel compilers + Intel’s MPI
module load intel-parallel-studio/cluster.2019.3-intelmpi
```

Compiling the code (depending on module and language):

```bash
# OpenMPI
mpicc -O2 -g hello_mpi.c -o hello_mpi  # C code
mpifort -O2 -g hello_mpi_fortran.f90 -o hello_mpi_fortran  # Fortran code

# Intel MPI
mpiicc -O2 -g hello_mpi.c -o hello_mpi  # C code
mpiifort -O2 -g hello_mpi_fortran.f90 -o hello_mpi_fortran  # Fortran code
```

Running the program with `srun` (for testing):

```bash
srun --time=00:05:00 --mem-per-cpu=200M --ntasks=4 ./hello_mpi
```

Running an MPI code in the batch mode:

```bash
#!/bin/bash
#SBATCH --time=00:05:00  # takes 5 minutes all together
#SBATCH --mem-per-cpu=200M  # 200MB per process
#SBATCH --ntasks=4  # 4 processes
#SBATCH --constraint=avx  # set constraint for processor architecture
module load openmpi/3.1.4  # NOTE: should be the same as you used to compile the code
srun ./hello_mpi
```

Triton has multiple architectures around (12, 20, 24, 40 CPU cores per node), even though SLURM optimizes resources usage and allocate CPUs within one node, which gives better performance for the app, it still makes sense to put constraints explicitly.

**Important:** It is important to use `srun` when you launch your program. This allows for the MPI libraries to obtain task placement information (nodes, number of tasks per node etc.) from the slurm queue.

**Spreading MPI workers evenly**

In many cases you might require more than one node during your job’s runtime.

When this is the case, it is usually recommended to split the number of workers somewhat evenly among the nodes. To do this, one can use `--N N/--nodes=N` and `--ntasks-per-node=n`. For example, the previous example could be written as:
#!/bin/bash
#SBATCH --time=00:05:00 # takes 5 minutes all together
#SBATCH --mem-per-cpu=200M # 200MB per process
#SBATCH --nodes=2 # 2 nodes
#SBATCH --ntasks-per-node=2 # 2 processes per node * 2 nodes = 4 processes in total
#SBATCH --constraint=avx # set constraint for processor architecture

module load openmpi/3.1.4
# NOTE: should be the same as you used to compile the code
srun ./hello_mpi

This way the number of workers is distributed more evenly, which in turn reduces communication overhead between workers.

Monitoring performance

You can use `seff <jobid>` to see what percent of available CPUs and RAM was utilized. Example output is given below:

```
$ seff 60985042
Job ID: 60985042
Cluster: triton
User/Group: tuomiss1/tuomiss1
State: COMPLETED (exit code 0)
Nodes: 1
Cores per node: 2
CPU Utilized: 00:00:29
CPU Efficiency: 90.62% of 00:00:32 core-walltime
Job Wall-clock time: 00:00:16
Memory Utilized: 1.59 MB
Memory Efficiency: 0.08% of 2.00 GB
```

If your processor usage is far below 100%, your code may not be working correctly. If your memory usage is far below 100% or above 100%, you might have a problem with your RAM requirements. You should set the RAM limit to be a bit above the RAM that you have utilized.

You can also monitor individual job steps by calling `seff` with the syntax `seff <jobid>.<job step>.

**Important:** When making job reservations it is important to distinguish between requirements for the whole job (such as --mem) and requirements for each individual task/cpu (such as --mem-per-cpu). E.g. requesting --mem-per-cpu=2G with --ntasks=2 and --cpus-per-task=4 will create a total memory reservation of (2 tasks)*(4 cpus / task)*(2GB / cpu)=16GB.

Exercises

The scripts you need for the following exercises can be found in this git repository: `hpc-examples`. You can clone the repository by running `git clone https://github.com/AaltoSciComp/hpc-examples.git`. This repository will be used for most of the tutorial exercises.

1. Run `srun --cpus-per-task=4 hostname`, `srun --ntasks=4 hostname` and `srun --nodes=4 hostname`. What’s the difference and why?

2. Find the files `hpc-examples/openmp/hello_omp/hello_omp.c` and `hpc-examples/openmp/hello_omp/hello_omp.slrm` that have a short example of OpenMP. Compile and run it - a slurm
3. Find the files in hpc-examples/python/python_openmp. Try running the example with a few different --constraint=X and --cpus-per-task=C. In your opinion, what architecture / cpu number combination would provide the best efficiency? Use seff to verify.

4. Find the files hpc-examples/mpi/hello_mpi/hello_mpi.c and hpc-examples/mpi/hello_mpi/hello_mpi.slrm that have a short example of MPI. Compile and run it - a slurm script is included.

Next steps

See the next pages:

- You can check the Running programs on Triton page for the reference information on running jobs. This contains the general reference information.

- MPI on Triton

Job dependencies

Introduction

Job dependencies are a way to specify dependencies between jobs. The most common use is to launch a job only after a previous job has completed successfully. But other kinds of dependencies are also possible.

Basic example

Dependencies are specified with the --dependency=<dependency list> option. E.g. --dependency=afterok:123:124 means that the job can only start after job ID’s 123 and 124 have both completed successfully.

Automating job dependencies

A common problem with job dependencies is that you want job B to start only after job A finishes successfully. However, you cannot know the job ID of job A before it has been submitted. One solution is to catch the job id of job A when submitting it and store it as a shell variable, and using the stored value when submitting job B. Like:

\[
\text{idA}=$(\text{sbatch jobA.sh | awk '{print $4}')}
\]

\[
\text{sbatch --dependency=afterok:${idA}} \text{ jobB.sh}
\]

Exercises

1. Look at man sbatch and investigate the --dependency parameter.

2. Create a chain of jobs A -> B -> C each depending on the successful completion of the previous job. In each job run e.g. sleep 60 to give you time to investigate the status of the queue.

3. Continuing from the previous exercise, what happens if at the end of the job A script you put exit 1. What does it mean?
6.1.4 Detailed instructions

Available compilers

Please see full up to date listing of different toolchains and compilers from Applications-page. This page contains information on their usage.

GCC

Triton has the GCC set of compilers installed by default, but we recommend that you use the provided module versions. The GNU Compiler Collection (aka GCC) includes front ends for C, C++, Objective-C, Fortran, Java, Ada, and Go, as well as libraries for these languages (libstdc++, libc++. . . ). In case of missing features, contact admins.

$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/share/apps/easybuild/software/GCCcore/5.4.0/libexec/gcc/x86_64-unknown-linux-gnu/5.4.0/lto-wrapp
Target: x86_64-unknown-linux-gnu
Configured with: ../configure --enable-languages=c,c++,fortran --enable-lto --enable-
--checking-release --disable-multilib --enable-shared=yes --enable-static=yes --
--enable-threads=posix --enable-gold=default --enable-plugins --enable-ld --with-
--plugin-ld=1d.gold --enable-bootstrap --prefix=/share/apps/easybuild/software/
--GCCcore/5.4.0 --with-local-prefix=/share/apps/easybuild/software/GCCcore/5.4.0
Thread model: posix
gcc-version 5.4.0 (GCC)

Example usage:

$ gcc -lm -o my_code.c my_code # compiling your C code and linking with Math lib
$ gfortran -O2 -o my_code my_code.f90 # compiling your Fortran code with -O2 --
--optimization level (see also g77 for Fortran 77)
$ g++ -O3 -funroll-loops -ffast-math -ftree-vectorize -mtune=native -o my_code my_
--code.cpp # compiling your C++ code with aggressive optimization and architecture
--tuning

Thus GCC is the default compiler and is used to build most of the software in the cluster.

See man gcc, man gfortran and other mans for options. Online GCC Documentation.

GCC compiling examples

$ cat hello.c
#include
int main(void) {
    printf("Hello World!\n");
    return 0;
}
$ gcc hello.c -o hello
Compiling your own code

- Use gcc, g++, and gfortran compilers for compilation
- Use mpicc, mpic++, and mpif90 for MPI (= MVAPICH, MPICH2 or OpenMPI + GCC)
- Setup your environment with module load <toolchain> e.g. module load goolf/triton-2016b for BLAS/LAPACK, FFTW3, ScaLAPACK+BLACS, etc.. Modules will set CPATH and LD_LIBRARY_PATH variables for -I and -L, but you can use 'module show <module>' to see the exact library paths.

MPI-code “mpihello.c”:

```c
#include
#include
int main (int argc, char **argv) {
    int rank, size;
    MPI_Init (&argc, &argv);
    printf( "Hello world\n");
    MPI_Finalize();
    return 0;
}
```

Compile MPI code:

```bash
module load goolf
mpicc mpihello.c -o mpihello
```

Optimization options for GCC

By default GCC/G++/GFortran do NOT perform any optimization; you must add appropriate optimization flags yourself. Experiment and see what works for your program!

- Basic optimization level: -O2
- More aggressive optimization, arch specific: -O3 -march=native
- Might, or might not help: -O3 -march=native -funroll-loops -ftree-loop-linear
- -fprefetch-loop-arrays

Might help a lot, but potentially dangerous: -ffast-math -mrecip

OpenMP with GCC: -fopenmp

Note that using -march=native will produce an arch specific code. Thus when compiled on Haswell, that code must be run on Haswell, otherwise expect segmentation fault errors on other architectures.

Intel Compilers

Intel Composer available through module provides full set of compilers C/C++/Fortran. In addition, it comes with MKL libraries, that works well in case you need LAPACK/BLAS functionality, as well as parallel version of them with ScaLAPACK and BLACS.

Example of linking for Intel Composer + MKL + OpenMPI

```bash
$ module load ioolf
$ mpif90 -o my.exe one.o two.o three.o libmpi_f90.a -lmkl_scalapack_lp64 -Wl,--start-
  group -lmkl_intel_lp64 -lmkl_sequential -lmkl_core -lmkl_blacs_openmpi_lp64 -Wl,--
  end-group -lpthread -lm
```

(continues on next page)
Compiling CUDA code while using conda environments

Conda is a powerful package manager that is commonly used to create Python environments. It is often used to install GPU-accelerated code such as PyTorch or Tensorflow. Many models built on top of these frameworks often extend the available operators / CUDA kernels by compiling extensions. These extensions are sometimes built beforehand and sometimes they are done using JIT (just-in-time) compilation. When dealing with such models one can often encounter many pitfalls that make it hard to compile of said extensions.

This document tries to explain how one should approach and debug CUDA compiling while using conda environments. The example uses PyTorch, but the basic ideas work for other frameworks as well.

TL;DR is provided at the end of the document.

Our example case

In this example we’ll use PyTorch’s example repo on C++/CUDA extensions.

```
git clone https://github.com/pytorch/extension-cpp.git pytorch-extension-cpp
cd pytorch-extension-cpp/cuda
module load miniconda
conda create --name pytorch-env --channel pytorch pytorch torchvision torchaudio
   --cudatoolkit=10.2
source activate pytorch-env
```

Here we:

1. Cloned the extension repository and moved into its CUDA-examples folder
2. Loaded a miniconda module that gives us the conda-command
3. Created a new environment for our pytorch installation and activated it

One could of course use the already existing anaconda-environment, but when dealing with extensions one often needs a specific version of a toolkit and/or framework, so here we were using a custom environment.

How conda packages the CUDA libraries

During the installation procedure you might have noticed that we obtained a package called cudatoolkit. In fact, during the environment creation we explicitly wanted a specific version of this toolkit (`cudatoolkit=10.2`). This requirement only specified the major version, so to see what is the full version of our toolkit, we need to run

```
conda list cudatoolkit
```

End result is something like this:
So the version we have installed is 10.2.89. This is important, as all packages installed by conda that use this toolkit have been compiled to use the specific version of the toolkit. If we run

```
conda list pytorch
```

We see something like this:

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>Build</th>
<th>Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>pytorch</td>
<td>1.8.1</td>
<td>py3.8_cuda10.2_cudnn7.6.5_0</td>
<td>pytorch</td>
</tr>
</tbody>
</table>

Here we can see that the version of our pytorch-package is 1.8.1, the build of the package is py3.8_cuda10.2_cudnn7.6.5_0 and it comes from a channel called pytorch. Looking at the build-string we can see that our version of pytorch has been compiled against CUDA 10.2 and cuDNN 7.6.5.

For more information on the build, we can run

```
conda search --channel pytorch --info pytorch=1.8.1=py3.8_cuda10.2_cudnn7.6.5_0
```

which will show all of the dependencies that the package has. For now, we’re only interested in the version of the cudatoolkit.

If you’re installing multiple different CUDA-enabled frameworks into a single environment it is recommended to do the installation in a single command as otherwise you might get competing builds with competing cudatoolkit-requirements. This can break some of your installations.

Why is the version of cudatoolkit so important? That is because the cudatoolkit that comes via conda is not the full CUDA SDK (software development kit). It is missing, among other things, the nvcc compiler that is used to compile CUDA code. The package cudatoolkit only contains runtime libraries, not development headers etc. It is done in this way to save space. Most use cases for CUDA code do not compile their own CUDA code and thus packaging the minimal amount of files to the toolkit will greatly reduce the network bandwidth and storage needed by environments.

However, when we’re compiling CUDA extensions, we need the CUDA compiler and the development headers. These we will find from the module system.

## Obtaining CUDA SDK from the module system

Our module system contains various installations of the CUDA toolkit. In order to use them properly, we also need to load a compatible compiler.

Let’s first try to run a JIT-compiled extension without loading the correct modules. We can (in the pytorch-extension-cpp/cuda-folder) try the JIT-compiled code on a GPU node.

```
srun --gres=gpu:1 --mem=4G --time=00:15:00 python jit.py
```

This will fail with error such as
RuntimeError: Error building extension 'lltm_cuda'

This happens because we are missing the required development files. To load said files we can run

```
module load gcc/8.4.0
module load cuda/10.2.89
```

Here we do the following:

1. We load a compiler that is supported by our version of CUDA-toolkit.
2. We load a CUDA SDK with exactly the same version as the one installed in our conda environment.

If we try our `srun`-command again after loading the modules, we get (after some compilation output) the following:

```
Loading extension module lltm_cuda...
Help on module lltm_cuda:

NAME
    lltm_cuda

FUNCTIONS
    backward(...) method of builtins.PyCapsule instance
        backward(arg0: at::Tensor, arg1: at::Tensor, arg2: at::Tensor, arg3:
                   at::Tensor, arg4: at::Tensor, arg5: at::Tensor, arg6: at::Tensor, arg7: at::Tensor,
                   arg8: at::Tensor) -> List[at::Tensor]

    LLTM backward (CUDA)

    forward(...) method of builtins.PyCapsule instance
        forward(arg0: at::Tensor, arg1: at::Tensor, arg2: at::Tensor, arg3:
                   at::Tensor, arg4: at::Tensor) -> List[at::Tensor]

    LLTM forward (CUDA)

FILE
    /scratch/work/tuomiss1/cache/torch_extensions/lltm_cuda/lltm_cuda.so
```

This means that our compilation was successful.

During these steps it is important to notice few things.

Firstly, different versions of CUDA only support a range of compilers. In case of CUDA 10.2, GCC 8.4.0 is within the supported range. To find out what are the supported versions for specific CUDA toolkit one needs to find out this table hidden in the CUDA toolkit’s installation requirements. It lists the minimum and maximum version numbers for the compiler. However, for modern versions of the CUDA toolkit the version used as a base compiler for Triton should be good enough (e.g. gcc/8.4.0 at the time of writing).

Secondly, it is recommended to exactly match the module version of the CUDA toolkit with version of the CUDA toolkit that is within the conda environment. If we’re missing a module version of CUDA toolkit that you have installed via conda, please let us know. Changes in second minor version might not affect the stability or results of a compiled program, but it is not worth the risk to try different versions. Installing various CUDA-toolkits as modules is very easy for us.

6.1. Triton cluster
TL;DR

1. Install cuda-enabled code in your conda environment and activate it
2. Find out what version of cudatoolkit was installed with
   
   ```
   conda list cudatoolkit
   ```
3. Load compiler and CUDA SDK with the same version with
   
   ```
   module load gcc/8.4.0
   module load cuda/"exact same version as in conda environment"
   ```

Debugging

**Note:** Also see *Profiling*.

Debugging is one of the most fundamental things you can do while using software: debuggers allow you to see inside of running programs, and this is a requirement of developing with any software. Any reasonable programming language will have a debugger made as one of the first tasks when it is being created.

**Serial code debugging**

**GDB** is the usual GNU debugger.

Note: the latest version of gcc/gfortran available through module require `-gdwarf-2` option along with the `-g` to get it to work with the default gdb command. Otherwise the default version 4.4 should work normally with just `-g`.

Valgrind is another tool that helps you to debug and profile your serial code on Triton.

**MPI debugging & profiling**

**GDB with the MPI code**

Compile your MPI app with `-g`, run GDB for every single MPI rank with:

```
  Alloc -p play --nodes 1 --ntasks 4 srun xterm -e gdb mpi_app
```

You should get 4 xterm windows to follow, from now on you have full control of you MPI app with the serial debugger.

**PADB**

A Parallel Debugging Tool. Works on top of SLURM, support OpenMPI or MPICH only (as of June 2015), that is MVAPICH2 is not supported. Do not require code re-compilation, just run your MPI code normally, and then launch padb separately to analyze the code behavior.

Usage summary (for full list and explanations please consult [http://padb.pittman.org.uk/](http://padb.pittman.org.uk/)):
Aalto scientific computing guide

```bash
# assume you have your openmpi module loaded already
module load padb
padb --create-secret-file  # for the very first time only

# Show all your current active jobs in the SLURM queue
padb -show-jobs

# Target a specific jobid, and reports its process state
padb --proc-summary
# or, for all running jobs
padb --all --proc-summary

# Target a specific jobid, and report its MPI message queue, stack traceback, etc.
padb --full-report=

# Target a specific jobid, and report its stack trace for a given MPI process (rank)
padb --stack-trace --tree --rank

# Target a specific jobid, and report its stack trace including information about
# parameters and local variables for a given MPI process (rank)
padb --stack-trace --tree --rank -Ostack-shows-locals=1 -Ostack-
# shows-params=1

# Target a specific jobid, and reports its MPI message queues
padb --mpi-queue

# Target a specific jobid, and report its MPI process progress (queries in loop over
# and over again)
padb --mpi-watch --watch -Owatch-clears-screen=no
```

Nvidia DGX machines

Triton currently has two Nvidia DGX-1 machines machines which contain 8 V100 GPUs and are optimized for deep learning.

**Warning:** The DGX usage in Slurm, and this page in general, are under development and testing. For latest changes, you can check git history using the link in the top right corner.

Access and prerequisites

The DGX machines have been specifically bought by several groups and these groups have priority access.

**General access:** you should us the `dgx-common` partition. This has preemption enabled, which means that if a higher priority job comes, you job can be cancelled at any time, even if it is running. The job will then be added back to the queue and possibly run again. Design your code to take this into account. Furthermore, your job will only start running when the priority partition is empty... so in effect jobs happen very slowly. If you are using general access, all of the `-p dgx` in the examples below need to be changed to `-p dgx-common`.

**Dedicated group access:** You can check the groups which may access it by running `grep PartitionName=dgx /etc/slurm/slurm.conf` and checking `AllowedGroups=`, check your groups with `groups`, and check all group members with `getent group $groupname`. If you should have access but don’t, **email our support alias** with a CC to your group leader, and we will fix this.

You also need a Triton account.

6.1. Triton cluster
Basics

The DGX machines have a special operating system from Nvidia based on Ubuntu 16.04, and thus form a very special of a Triton node because the rest of Triton is CentOS. We have done work to make them work together, but it will require special effort to make code run on both halves. You may find some problems, so please be aggressive about filing issues (but also aggressive about checking the background yourself and giving us good information).

Basic reading: Connecting to Triton.

Unlike before, direct access is not available: you should connect to the login node and submit jobs via Slurm, not running directly interactively.

Software and modules

Basic reading: Software modules.

You should load software using the module command, just like the rest of Triton. However, since the base operating system is different, modules are not automatically compatible. So, you can’t automatically reuse the modules you use on the rest of Triton.

The current available modules are:

```
--------- /share/apps/anaconda-ci/modules ---------
  anaconda2/5.1.0-cpu  anaconda3/5.1.0-cpu
  anaconda2/5.1.0-gpu (D)  anaconda3/5.1.0-gpu (D)
--------- /share/apps/singularity-ci/dgx/modules/common ---------
  nvidia-caffe/18.02-py2  nvidia-pytorch/18.11-py3 (D)
  nvidia-cntk/18.02-py3  nvidia-tensorflow/18.02-py2
  nvidia-mxnet/18.02-py2  nvidia-tensorflow/18.02-py3
  nvidia-mxnet/18.03-py3  nvidia-tensorflow/19.01-py3 (D)
  nvidia-mxnet/18.11-py3 (D)  nvidia-theano/18.02
  nvidia-pytorch/18.02-py3  nvidia-torch/18.02-py2
  nvidia-pytorch/18.08-py3  singularity-tensorflow/latest
--------- /share/apps/modulefiles/dgx ---------
  matlab/r2012a  matlab/r2015b  matlab/r2016b  matlab/r2018a
  matlab/r2014a  matlab/r2016a  matlab/r2017b  matlab/r2018b (D)
--------- /usr/share/lmod/lmod/modulefiles/Core ---------
  lmod/6.6  settarg/6.6
```

Unlike the rest of Triton, you can’t see which modules are available on the login node: currently see above (which might go out of date) or get an interactive shell on the DGX node (see below) and run module avail yourself.
Running jobs

Basic reading: tutorials on interactive jobs, serial jobs

All runs on the DGX machines go via Slurm. For an introduction to slurm, see the tutorials linked above, and in general all the rest of the Triton user guide. Slurm is a cluster scheduling system, which takes job requests (code, CPU/memory/time/hardware requirements) and distributes it to nodes. You basically need to declare what your jobs require, and tell it to run on DGX nodes.

Basic required slurm options

The necessary Slurm parameters are:

- \(-p \ dgx\) (dedicated group access) or \(-p \ dgx\)-common (general access, jobs may be killed at any time, see above) to indicate that we want to run in the DGX partitions.
- \(--gres=gpu:v100:1\) to request GPUs (Slurm also manages GPUs and limits you to the proper devices).
  - To request more than one graphics card, \(--gres=gpu:v100:2\)
- \(--export=HOME,USER,TERM,WRKDIR\) to limit the environment exported. Because these are a different operating system, you need to clear most environment variables. If there are extra environment variables you need, add them here.
- \(/bin/bash -l\): you need to give the full path to bash and request a login shell, or else the environment won’t be properly set by Slurm.
- \(--time=HH:MM:SS\) To set the run time. \(--time=HH:MM:SS\) If you want more CPUs, add \(--c\ \ N\). If you want more (system) memory, use \(--mem=5GB\) and so on. (These are completely generic slurm options.)

To check running and jobs: \(squeue -p dgx, dgx-common\) (whole cluster) or \(slurm q\) (for your own jobs).

Getting an interactive shell for own work

For example, to get an interactive shell, run:

```bash
srun -p dgx --gres=gpu:v100:1 --export=HOME,USER,TERM,WRKDIR --pty /bin/bash -l
```

From here, you can do whatever you want interactively with your dedicated resources almost as if you logged in directly. Remember to log out when done, otherwise your resources stay dedicated to you and no one else can use them!

Batch scripts

Similarly to the rest of Triton, you can make batch scripts:

```
#!/bin/bash -l
#SBATCH -p dgx
#SBATCH --time=00:05:00
#SBATCH --mem=5G
#SBATCH --gres=gpu:1
#SBATCH --export=HOME,USER,TERM,WRKDIR

your shell commands here
```
Nvidia containers

Some of the Nvidia containers designed for the DGX machines are available as modules - see above. They are integrated with our Triton singularity setup, so you can use those same procedures:

```bash
module load nvidia-tensorflow

# Get a shell within the image:
singularity_wrapper shell

# Execute Python within the image
singularity_wrapper exec python3 code.py
```

singularity_wrapper sets the image file (from the module you loaded), important options (to bind-mount things), and starts it.

This is a minimum slurm script (submit with `sbatch`, see the slurm info above and tutorials for more info):

```bash
#!/bin/bash -l
#SBATCH -p dgx
#SBATCH --time=00:05:00
#SBATCH --mem=5G
#SBATCH --gres=gpu:1
#SBATCH --export=HOME,USER,TERM,WRKDIR

module load nvidia-tensorflow
singularity_wrapper exec python -V
```

Other notes

Note: if you are using tensorboard, just have it write data to the scratch filesystem, mount that on your workstation, and follow it that way. See the data storage tutorial.

Within jobs, use `/tmp` for temporary local files. This is bind-mounted per user (not per job, make sure that you prefix by job ID or something to not get conflicts) to the `/raid` SSD area. (note: see below, this doesn’t work yet)

Known bugs

- You have to give the full path to `/bin/bash` and give the `--l` option to make a login shell to read necessary shell initialization.
- You have to limit the environment variables you export, because they are different. But you have to export at least `HOME` and possibly more (see above).
- You can’t figure out modules are available without getting an interactive shell there.
- The `/tmp` directory is not automatically to a per-user tmpdir (or `/raid`). For large amounts of intermediates, use a per-user subdirectory of `/raid` for your work.
- `/scratch` isn’t automatically mounted for some reason. For now, we manually mount it on each reboot but this needs fixing.
Frequently asked questions

Frequently Asked Questions. The latest are at the beginning.

**libcuda.so.1: cannot open shared object file: No such file or directory**

You are trying to run a GPU program (using CUDA) on a node without a GPU (and thus, no libcuda.so.1. Remember to specify that you need GPUs.

**Why are my jobs waiting in the queue with reason AssocGrpMemRunMinutes/AssocGrpCPURunMinutes or such**

Accounts are limited in how much the can run at a time, in order to prevent a single or a few users from hogging the entire cluster with long-running jobs if it happens to be idle (e.g. after a service break). The limit is such that it limits the maximum remaining runtime of all the jobs of a user. So the way to run more jobs concurrently is to run shorter and/or smaller (less CPU’s, less memory) jobs. For an in-depth explanation see [http://tech.ryancox.net/2014/04/scheduler-limit-remaining-cputime-per.html](http://tech.ryancox.net/2014/04/scheduler-limit-remaining-cputime-per.html) and for a graphical simulator you can play around with: [https://rc.byu.edu/simulation/grcrrunmins.php](https://rc.byu.edu/simulation/grcrrunmins.php). You can see the exact limits of your account with

```
sacctmgr -s show user $USER format=user,account,grptresrunmins%70
```

**Why are my jobs in state “launch failed requeued held”**

Slurm is configured such that if a job fails due to some outside reason (e.g. the node where it’s running fails rather than the job itself crashing due to a bug in the job) the job is requeued in a held state. If you’re sure that everything is ok again you can release the job for scheduling with “scontrol release JOBID”. If you don’t want this behavior (i.e. you’d prefer that such failed jobs would just disappear) then you can prevent the requeuing with

```
#SBATCH --no-requeue
```

**Invalid account . . . error message**

While submitting a job you receive an error message like

```
sbatch: error: Batch job submission failed: Invalid account or account/partition combination specified
```

Most probably your account is missing from SLURM database, to check it out run

```
$ sacctmgr show user $USER format=Def Acct Admin

User   Def Acct   Admin
---------  ---------  ---------
YOUR_LOGIN YOUR_DEPART None
```

That should return your login and associated department/school. If empty, please contact your local support team member and ask to add your account to SLURM db.
How can I find out the remaining runtime of my job/allocation

You can find out the remaining time of any job that is running with

```
squeue -h -j -o %L
```

Inside a job script or `sinteractive` session you can use the environment variable SLURM_JOB_ID to refer to the current job ID.

Disk quota exceeded error but I have plenty of space

Main article: *Triton Quotas*

Everyone should have a group quota, but no user quota. All files need to be in a proper group (either a shared group with quota, or your “user private group”). First of all, use the ‘quota’ command to make sure that neither disk space nor number of files are exceeded. Also, make sure that you use `$WRKDIR` for data and not `$HOME`. If you actually need more quota, ask us.

Solution: add to your main directory and all your subdirectories to the right group, and make sure all directories have the group s-bit set, (`SETGID` bit, see `man chmod`). This means “any files created within this directory get the directory’s group”. Since your default group is “domain users” which has no quota, if the s-bit is not set, you get an immediate quota exceeded by default.

```
# Fix everything
# (only for $WRKDIR or group directories, still in testing):
/share/apps/bin/quotafix -sg --fix /path/to/dir/

# Manual fixing:
# Fix sticky bit:
lfs find $WRKDIR -type d --print0 | xargs -0 chmod g+s
# Fix group:
lfs find /path/to/dir ! --group $GROUP -print0 | xargs -0 chgrp $GROUP
```

Why this happens: `$WRKDIR` directory is owned by the user and user’s group that has the same name and GID as UID. Quota is set per group, not per user. That is how it was implemented since 2011 when we got Lustre in use. Since spring 2015 Triton is using Aalto AD for the authentication which sets everyone a default group ID to ‘domain users’. If you copy anything to `$WRKDIR/subdirectory` that has no +s bit you copy as a ‘domain users’ member and file system refuses to do so due to no quota available. If g+s bit is set, all your directories/files copied/created will get the directory’s group ownership instead of that default group ‘domain users’. There can be very confusing interactions between this and user/shared directories.

My `$WRKDIR` is not visible on my department computer

Most likely your Kerberos ticket has expired. If you log in with a password or use ‘kinit’, you can get an another ticket. See page on *data storage* for more information.
While copying to $WRKDIR with rsync or cp I’m getting ‘Disk quota exceeded’ error, though my quota is fine.

It is related to the above mentioned issue, something like rsync -a ... or cp -p ... are trying to save original group ownership attribute, which will not work. Try this instead:

```
## mainly one should avoid -g (as well as -a) that preserves group attributes
$ rsync -urlptDxv --chmod=Dg+s somefile triton.aalto.fi:/path/to/work/directory
## avoid '-p' with cp, or if you want to keep timesteps, mode etc, then use '--
˓→-preserve=

$ cp -r --preserve=mode,timestamps somefile /path/to/mounted/triton/work/directory
```

Can I change zsh to bash?

Yes. Change shell to your Aalto account and re-login to Triton to get your newly changed shell to work. For Aalto account changes one can login to kosh.aalto.fi, run kinit first and then run chsh, then type /bin/bash. To find out what is your current shell, run echo $SHELL

For the record: your default shell is not set by Triton environment but by your Aalto account.

Job fails due to missed module environment variables.

You have included ‘module load module/name’ but job still fails due to missing shared libraries or that it can not find some binary etc. That is a known ZSH related issue. In your sbatch script please use --l option (aka --login) which forces bash to read all the initialization files at /etc/profile.

```
#!/bin/bash -l
...
```

Alternatively, one can change shell from ZSH to BASH to avoid this hacks, see the post above.

There seems to be running a lot of jobs in the short queue that has gone for longer than 4 hours. Should that be possible?

SLURM kills jobs based on the partition’s TimeLimit + OverTimeLimit parameter. The later in our case is 60 minutes. If for instance queue time limit is 4 hours, SLURM will allow to run on it 4 hours, plus 1 hour, thus no longer than 5 hours. Though OverTimeLimit may vary, don’t rely on it. Partition’s (aka queue’s) TimeLimit is the one that end user should take into account when submit his/her job. Time limits per partition one can check with slurm p command.

For setting up exact time frame after which you want your job to be killed anyway, set --time parameter when submitting the job. When the time limit is reached, each task in each job step is sent SIGTERM followed by SIGKILL. If you run a parallel job, set --time with srun as well. See ‘man srun’ and ‘man sbatch’ for details.

```
#SBATCH --time=1:00:00
...
```
How can I print my text file to a local department printer?

We don’t have local department printers configured anywhere on Triton. But one can use SSH magic to send a file or command output to a remote printer. Run from your local workstation, insert the target printer name:

```
... printing text file
$ ssh user@triton.aalto.fi "cat file.txt" | enscript -P printer_name
... printing a PostScript file
$ ssh user@triton.aalto.fi "cat file.ps" | lp -d printer_name -
... printing a man page
$ ssh user@triton.aalto.fi "man -t sbatch" | lp -d printer_name -
```

How can I access my Triton files from outside?

Remote mounting

The scratch filesystem can be mounted from inside the Aalto networks by using `smb://data.triton.aalto.fi/scratch/`. For example, from Nautilus (the file manager) on Ubuntu, use “File” -> “Connect to server”. Outside Aalto networks, use the Aalto VPN. If it is not an Aalto computer, you may need to use AALTO\username as the username, and your Aalto password.

Or you can use `sshfs` – filesystem client based on SSH. Most Linux workstations have it installed by default, if not, install it or ask your local IT support to do it for you. For setting up your SSHFS mount from your local workstation: create a local directory and mount remote directory with sshfs

```
$ mkdir /LOCALDIR/triton
$ sshfs user1@triton.aalto.fi:/triton/PATH/TO/DIR /LOCALDIR/triton
```

Replace `user1` with your real username and `/LOCALDIR` with a real directory on your local drive. After successful mount, use you `/LOCALDIR /triton` directory as it would be local. To unmount it, run `fusermount -u /LOCALDIR/triton`

**PHYS users example, assuming that Triton and PHYS accounts are the same:**

```
$ mkdir /localwrk/$USER/triton
$ sshfs triton.aalto.fi:/triton/PATH/TO/DIR /LOCALDIR/triton
$ cd /localwrk/$USER/triton
... (do what you need, and then unmount when there is no need any more)
$ fusermount -u /localwrk/$USER/triton
```

Easy access with Nautilus

The SSHFS method described above works from any console. Though in case of Linux desktops, when one has a GUI like Gnome or Unity (read all Ubuntu users) one may use Nautilus – default file manager – to mount remote SSH directory. Click File -> Connect to Server choose SSH, input triton.aalto.fi as a server and directory `/triton/PATH/TO/DIR` you’d like to mount, type your name. Leave password field empty if you use SSH key. As soon as Nautilus will establish connection it will appear on the left-hand side below Network header. Now you may access it as it would be your local directory. To keep it as a bookmark click on the mount point and press Ctrl+D, it will appear below Bookmark header on the same menu.
Copying files

If your workstation has no NFS mounts from Triton (CS and NBE have, consult with your local admins for exact paths), you may always use SSH. Either copy your files from triton to a local directory on your workstation, like:

```
$ sftp user1@triton.aalto.fi:/triton/path/to/dir/* .
```

**How can I copy Triton files from outside of Aalto?**

It is an extension of the previous question. In case you are outside of Aalto and has neither direct access to Triton nor access to NFS mounted directories on your directory servers. Say you want to copy your Triton files to your home workstation. It could be done by setting up an SSH tunnel to your department SSH server. A few steps to be done: set tunnel to your local department server, then from your department server to Triton, and then run any rsync/sftp/ssh command you want from your client using that tunnel. The tunnel should be up during whole session.

```
client: ssh -L9509:localhost:9509 department.ssh.server
department server: ssh -L9509:localhost:22 triton.aalto.fi
client: sftp -P 9509 localhost:/triton/own/dir/* /local/dir
```

Note that port 9509 is taken for example only. One can use any other available port. Alternatively, if you have a Linux or Mac OS X machine, you can setup a “proxy command”, so you don’t have to do the steps above manually everytime. On your home machine/laptop, in the file ~/.ssh/config put the lines

```
Host triton
  ProxyCommand /usr/bin/ssh DEPARTMENTUSERNAME@department.ssh.server "'/usr/bin/nc -w 10 triton.aalto.fi 22"
  User TRITONUSERNAME
```

This creates a host alias “triton” that is proxied via the department server. So you can copy a file from your home machine/laptop to triton with a command like:

```
rsync filename triton:remote_filename
```

**I need to connect to some server on a node**

Let’s say you have some server (e.g. debugging server, notebook server, ...) running on a node.

**Why are my jobs waiting in the queue with reason AssocGrpMemRunMinutes/AssocGrpCPURunMinutes or such**

Accounts are limited in how much the can run at a time, in order to prevent a single or a few users from hogging the entire cluster with long-running jobs if it happens to be idle (e.g. after a service break). The limit is such that it limits the maximum remaining runtime of all the jobs of a user. So the way to run more jobs concurrently is to run shorter and/or smaller (less CPU’s, less memory) jobs. For an in-depth explanation see [http://tech.ryancox.net/2014/04/scheduler-limit-remaining-cpu-time-per.html](http://tech.ryancox.net/2014/04/scheduler-limit-remaining-cpu-time-per.html) and for a graphical simulator you can play around with: [https://rc.byu.edu/simulation/grpctpurunmins.php](https://rc.byu.edu/simulation/grpctpurunmins.php). You can see the exact limits of your account with

```
sacctmgr -s show user $USER format=user,account,grptresrunmins%70
```
Why are my jobs in state “launch failed requeued held”

Slurm is configured such that if a job fails due to some outside reason (e.g. the node where it’s running fails rather than the job itself crashing due to a bug in the job) the job is requeued in a held state. If you’re sure that everything is ok again you can release the job for scheduling with “scontrol release JOBID”. If you don’t want this behavior (i.e. you’d prefer that such failed jobs would just disappear) then you can prevent the requeuing with

```
#SBATCH --no-requeue
```

Invalid account ... error message

While submitting a job you receive an error message like

```
sbatch: error: Batch job submission failed: Invalid account or account/partition combination specified
```

Most probably your account is missing from SLURM database, to check it out run

```
$ sacctmgr show user $USER
```

That should return your login and associated department/school. If empty, please contact your local support team member and ask to add your account to SLURM db.

How can I find out the remaining runtime of my job/allocation

You can find out the remaining time of any job that is running with

```
squeue -h -j -o %L
```

Inside a job script or sinteractive session you can use the environment variable SLURM_JOB_ID to refer to the current job ID.

Disk quota exceeded error

Main article: `Triton Quotas <quotas>`

```
space exceeded but I have plenty of space
```

Everyone should have a group quota, but no user quota. All files need to be in a proper group (either a shared group with quota, or your “user private group”). First of all, use the ‘quota’ command to make sure that neither disk space nor number of files are exceeded. Also, make sure that you use $WRKDIR for data and not $HOME. If you actually need more quota, ask us.

Solution: add to your main directory and all your subdirectories to the right group, and make sure all directories have the group s-bit set, (SETGID bit, see `man chmod`). This means “any files created within this directory get the directory’s group”. Since your default group is “domain users” which has no quota, if the s-bit is not set, you get an immediate quota exceeded by default.
# Fix everything
# (only for $WRKDIR or group directories, still in testing):
/share/apps/bin/quotafix -sg --fix /path/to/dir/

# Manual fixing:
# Fix sticky bit:
lfs find $WRKDIR -type d --print0 | xargs -0 chmod g+s
# Fix group:
lfs find /path/to/dir ! --group $GROUP -print0 | xargs -0 chgrp $GROUP

Why this happens: $WRKDIR directory is owned by the user and user’s group that has the same name and GID as UID. Quota is set per group, not per user. That is how it was implemented since 2011 when we got Lustre in use. Since spring 2015 Triton is using Aalto AD for the authentication which sets everyone a default group ID to ‘domain users’. If you copy anything to $WRKDIR/subdirectory that has no +s bit you copy as a ‘domain users’ member and file system refuses to do so due to no quota available. If g+s bit is set, all your directories/files copied/created will get the directory’s group ownership instead of that default group ‘domain users’. There can be very confusing interactions between this and user/shared directories.

While copying to $WRKDIR with rsync or cp I’m getting ‘Disk quota exceeded’ error, though my quota is fine.

It is related to the above mentioned issue, something like rsync -a ... or cp -p ... are trying to save original group ownership attribute, which will not work. Try this instead:

```bash
## mainly one should avoid -g (as well as -a) that preserves group attributes
$ rsync -urlptDxv --chmod=Dg+s somefile triton.aalto.fi:/path/to/work/directory

## avoid '-p' with cp, or if you want to keep timestamp, mode etc, then use '---preserve='
$ cp -r --preserve=mode,timestamps somefile /path/to/mounted/triton/work/directory
```

Can I change zsh to bash?

Yes. Change shell to your Aalto account and re-login to Triton to get your newly changed shell to work. For Aalto account changes one can login to kosh.aalto.fi, run kinit first and then run chsh, then type /bin/bash. To find out what is your current shell, run echo $SHELL

For the record: your default shell is not set by Triton environment but by your Aalto account.

Job fails due to missed module environment variables.

You have included ‘module load module/name’ but job still fails due to missing shared libraries or that it can not find some binary etc. That is a known ZSH related issue. In your sbatch script please use -l option (aka --login) which forces bash to read all the initialization files at /etc/profile.

```bash
#!/bin/bash -l
...
```

Alternatively, one can change shell from ZSH to BASH to avoid this hacks, see the post above.

6.1. Triton cluster
There seems to be running a lot of jobs in the short queue that has gone for longer than 4 hours. Should that be possible?

SLURM kills jobs based on the partition’s TimeLimit + OverTimeLimit parameter. The later in our case is 60 minutes. If for instance queue time limit is 4 hours, SLURM will allow to run on it 4 hours, plus 1 hour, thus no longer than 5 hours. Though OverTimeLimit may vary, don’t rely on it. Partition’s (aka queue’s) TimeLimit is the one that end user should take into account when submit his/her job. Time limits per partition one can check with `slurm p` command.

For setting up exact time frame after which you want your job to be killed anyway, set `--time` parameter when submitting the job. When the time limit is reached, each task in each job step is sent SIGTERM followed by SIGKILL.

```
#SBATCH --time=1:00:00
...

crun --time=1:00:00 ...
```

How can I print my text file to a local department printer?

We don’t have local department printers configured anywhere on Triton. But one can use SSH magic to send a file or command output to a remote printer. Run from your local workstation, insert the target printer name:

```
... printing text file
ssh user@triton.aalto.fi "cat file.txt" | enscript -P printer_name
... printing a PostScript file
ssh user@triton.aalto.fi "cat file.ps" | lp -d printer_name -
... printing a man page
ssh user@triton.aalto.fi "man -t sbatch" | lp -d printer_name -
```

How can I access my Triton files from outside?

If your workstations has no NFS mounts from Triton (CS and NBE have, consult with your local admins for exact paths), you may always use SSH. Either copy your files from triton to a local directory on your workstation, like

```
$ rsync -pr user1@triton.aalto.fi:/triton/path/to/dir .
```

or use SSHFS – filesystem client based on SSH. Most Linux workstations have it installed by default, if not, install it or ask your local IT support to do it for you. For setting up your SSHFS mount from your local workstation: create a local directory and mount remote directory with sshfs

```
$ mkdir /LOCALDIR/triton
$ sshfs user1@triton.aalto.fi:/triton/PATH/TO/DIR /LOCALDIR/triton
```

Replace `user1` with your real username and `/LOCALDIR` with a real directory on your local drive. After successful mount, use you `/LOCALDIR/triton` directory as it would be local. To unmount it, run `fusermount -u /LOCALDIR/triton`.  

PHYS users example, assuming that Triton and PHYS accounts are the same:

```
$ mkdir /localwrk/$USER/triton
$ sshfs triton.aalto.fi:/triton/TFY/work/$USER /localwrk/$USER/triton
$ cd /localwrk/$USER/triton
... (do what you need, and then unmount when there is no need any more)
$ fusermount -u /localwrk/$USER/triton
```
Easy access with Nautilus

The SSHFS method described above works from any console. Though in case of Linux desktops, when one has a GUI like Gnome or Unity (read all Ubuntu users) one may use Nautilus – default file manager – to mount remote SSH directory. Click File -> Connect to Server choose SSH, input triton.aalto.fi as a server and directory /triton/PATH/TO/DIR you’d like to mount, type your name. Leave password field empty if you use SSH key. As soon as Nautilus will establish connection it will appear on the left-hand side below Network header. Now you may access it as it would be your local directory. To keep it as a bookmark click on the mount point and press Ctrl+D, it will appear below Bookmark header on the same menu.

How can I copy Triton files from outside of Aalto?

It is an extension of the previous question. In case you are outside of Aalto and has neither direct access to Triton nor access to NFS mounted directories on your directory servers. Say you want to copy your Triton files to your home workstation. It could be done by setting up an SSH tunnel to your department SSH server. A few steps to be done: set tunnel to your local department server, then from your department server to Triton, and then run any rsync/sftp/ssh command you want from your client using that tunnel. The tunnel should be up during whole session.

```
client: ssh -L 9509:localhost:9509 department.ssh.server
department server: ssh -L 9509:localhost:22 triton.aalto.fi
client: sftp -P 9509 localhost:/triton/own/dir/* /local/dir
```

Note that port 9509 is taken for example only. One can use any other available port. Alteratively, if you have a Linux or Mac OS X machine, you can setup a “proxy command”, so you don’t have to do the steps above manually everytime. On your home machine/laptop, in the file ~/.ssh/config put the lines

```
Host triton
    ProxyCommand /usr/bin/ssh DEPARTMENTUSERNAME@department.ssh.server "/usr/bin/nc -w 10 triton.aalto.fi 22"
    User TRITONUSERNAME
```

This creates a host alias “triton” that is proxied via the department server. So you can copy a file from your home machine/laptop to triton with a command like:

```
rsync filename triton:remote_filename
```

I need to connect to some server on a node

Let’s say you have some server (e.g. debugging server, notebook server, ...) running on a node. As usual, you can do this with ssh using port forwarding. It is the same principle as in several of the above questions.

For example, you want to connect from your own computer to port AAAA on node nnnNNN. You run this command:

```
ssh -L BBBB:nnnNNN:AAAA username@triton.aalto.fi
```

Then, when you connect to port BBBB on your own computer (localhost, it gets forwarded straight to port AAAA on node nnnNNN. Thus only one ssh connection gets us to any node. It is possible for BBBB to be the same as AAAA. By the way, this works with any type of connection. The node has to be listening on any interface, not just the local interface. To connect to localhost:AAAA on a node, you need to repeat the above steps twice to forward from workstation->login and login->node, with the second nnnNNN being localhost.

6.1. Triton cluster
Why all of the files on triton cluster are in one color? How can I make them colorful? Like green for execution files, blue for folds

That is made intentionally due to high load on Lustre filesystem. Being a high performance filesystem Lustre still has its own bottlenecks, and one of the common Lustre troublemakers are `ls -lr` or `ls --color` which generate lots of requests to Lustre meta servers which regular usage by all users may get whole system in stuck. Please follow the recommendations given at the last section at *Data storage on the Lustre file system*.

How do I subscribe to triton-users maillist?

Having a user account on Triton also means being on the triton-users at aalto.fi maillist. That is where support team sends all the Triton related announcements. All the Triton users MUST be subscribed to the list. It is automatically kept up to date these days, but just in case you are not yet there, please send an email to your local team member and ask to add your email.

How to unsubscribe? You will be removed from the maillist as soon as your Triton account is deleted from the system. Otherwise no way, since we can’t notify about urgent things that affect data integrity or other issues.

I can’t save anything to my `$HOME` directory, get some fsync error.

Most probably your quota has exceeded, check it out with `quota` command.

`quota` is a wrapper at `/usr/local/bin/quota` on front end which merges output from classic quota utility that supports NFS and Lustre's `lfs quota`. NFS `$HOME` directory is limited to 10GB for everyone and intended for initialization files mainly. Grace period is set to 7 days and “hard” quota is set to 11GB, which means you may exceed your 10GB quota by 1GB and have 7 days to go below 10GB again. However none can exceed 11GB limit.

Note: Lustre mounted under `/triton` is the right place for your simulation files. It is fast and has large quotas.

What node names like cn[01-224] mean?

All the hardware delivered by the vendor has been labeled with some short name. In particular every single compute node has a label like Cn01 or GPU001 etc. we used this notation to name compute nodes, that is cn01 is just a hostname for Cn01, gpu001 is a hostname for GPU001 etc. Shorthands like cn[01-224] mean all the hostnames in the range cn01, cn02, cn03 .. cn224. Same for gpu[001-008], tb[003-008], fn[01-02]. Similar notations can be used with SLURM commands like:

```
$ scontrol show node cn[01-12]
```

What is a good scaling factor for parallel applications? What is the recommended number of processors for parallel jobs?

The good scaling factor is 1.5 or higher. It means that your program is running 1.5 times faster when you double the number of nodes.

There is no way to know in advance the exact “universal” optimal number of CPUs. It depends on many factors, like the application itself, type of MPI libraries, the initial input, I/O volume and the current network state. Certainly, you must not expect that, as many CPUs your application has got, that faster it will run. In general the scaling on Triton is good since we have Infiniband for nodes interconnect and DDN / Lustre for I/O.

Few recommendations about CPU number:
• benchmark your applications on different number of CPU cores 1, 2, 12, 24, 36, and larger. Check out with the developers, your application may have ready scalability benchmarks and recommendations for compiler, MPI libraries choice.

• benchmark on shared memory i.e. up to 12 CPU cores within one node and then on different nodes (distributed memory): involving interconnect make have huge difference

• if you are not sure about program scalability and you have no time for testing, don’t run on more than 12 CPU cores within one node

• be considerate! it is not you against others! do not try to fill up the cluster just for being cool

Can you recovery some files from my $HOME or $WRKDIR directory?

Short answer: yes for $HOME directory and no for $WRKDIR.

$HOME is slow NFS with small quota mounted through Ethernet. Intended mainly for user initialization files and for some plain configs. We make regular backups from $HOME.

$WRKDIR (aka /triton)is fast Lustre, has large quota, mounted through InfiniBand. Though no backups made from /triton, the DDN storage system as such is secure and safe place for your data, though you can always loose your data deleting them by mistake. Every user must take care about his work files himself. We provide as much diskspace to every user, as one needs and the amount of data is growing rapidly. That is the reason why the user should manage his important data himself. Consider backups of your valuable data on DVDs/ USB drives or other resources outside of Triton.

The cluster has a few compiler sets. Which one I suppose to use? What are the limits for commercial compilers?

Currently there are two different sets of compilers: (i) GNU compilers, native for Linux, installed by default, (ii) Intel compilers plus MKL, a commercial suite, often the fastest compiler on Xeons.

FGI provides all FGI sites with 7 Intel licenses, thus only 7 users can compile/link with Intel at once.

Code is compiled with shared libraries and it stops with an error message: error while loading shared libraries: libsome.so: cannot open shared object file: No such file or directory

That means your program can’t find libraries which has been used at linking/compiling time. You may always check shared library dependencies:

```
$ ldd YOUR_PROGRAM # print the list of libraries required by program
```

If some of libraries is marked as not found, then you should first (i) find the exact path to that lib (suppose it is installed), then second (ii) explicitly add it to your environment variable SLD_LIBRARY_PATH.

For instance, if your code has been previously compiled with the libmpi.so.0 but on SL6.2 it reports an error like error while loading shared libraries: libmpi.so.0 try to locate the library:
$ locate libmpi.so.0
/usr/lib64/compat-openmpi/lib/libmpi.so.0
/usr/lib64/compat-openmpi/lib/libmpi.so.0.0.2

and the add it to your $LD_LIBRARY_PATH

export LD_LIBRARY_PATH=/usr/lib64/compat-openmpi/lib:$LD_LIBRARY_PATH # export the lib

or, as in case of libmpi.so.0 we have ready module config, just run

module load compat-openmpi-x86_64

In case your code is missing some specific libs, not installed on Triton (say you got a binary compiled from somewhere else), you have a few choices: (i) get statically linked program or (ii) find/download missing libs (for instance from developers’ site). For the second, copy libs to your $WRKDIR and add paths to $LD_LIBRARY_PATH, in the same manner as described above.

See also:

ldconfig -p # print the list of system-wide available shared libraries

**While compiling should I use static or shared version of some library?**

One can use both, though for shared libs all your linked libs must be either in your $WRKDIR in /shared/apps or must be installed by default on all the compute nodes like vast majority of GCC and other default Linux libs.

**I’ve got a binary file, may I find out somehow whether it is 32-bit or 64-bit compiled?**

Use file utility:

```
# file /usr/bin/gcc
/usr/bin/gcc: ELF 64-bit LSB executable, AMD x86-64, version 1 (SYSV),
for GNU/Linux 2.4.0, dynamically linked (uses shared libs), not stripped
```

it displays the type of an executable or object file.

**Graphical programs don’t work (X11, -X)**

In order for graphical programs on Linux to work, a file ~/.Xauthority has to be written. If your home directory quota (check with quota) is exceeded, then this can’t be written and graphical programs can’t open. If your quota is exceeded, clean up some files, close connections, and log in again. You can find where most of your space goes with du -h $HOME | sort -hr | less.

This is often the case if you get X11 connection rejected because of wrong authentication.
Can I use a more up-to-date version of git on triton?

Indeed the default git with Triton OS system (CentOS) is quite old (v 1.8.x). To get a more modern git you can run module load git (version 2.28.0 when this is being written).

Running programs on Triton

Triton differs somewhat from your regular desktop computer. The large numbers and complex examples may give the impression of something far more special than it actually is: a bunch of computers. Fundamentally these are just slightly more powerful computers, with much more memory and a faster network in between. Where the major differences begin, though, is that they are shared by around 100 people from different departments with an unusual scale and variation of applications and needs. In order to even begin to accommodate everyone on the cluster, we have to use an intermediate resource manager and scheduler through which certain policies can be put into effect. The cluster is a combination of the compute nodes, our site policies, and the scheduler software which works it all out in practice.

This guide tries to give an idea of how to run programs in the cluster through the Slurm scheduler. While this certainly does not cover all the use cases, you’re welcome to ask any questions in the Issue Tracker.

Scheduling policy and queues

The cluster nodes (computers) are grouped into partitions (the scheduler’s concept). While the default batch partition may always be in full use, other partitions act as boundaries that keep specialized nodes, such as the GPU machines, ready and immediately available for jobs with special requirements.

<table>
<thead>
<tr>
<th>Partition</th>
<th>Max job size</th>
<th>Mem/core (GB)</th>
<th>Total mem (GB)</th>
<th>Cores/Node Limits</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;default&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>debug</td>
<td>2 nodes</td>
<td>2.66 - 12</td>
<td>32-256</td>
<td>12,20,24</td>
<td>15 min testing and debugging short interactive work, 1 node of each arch.</td>
</tr>
<tr>
<td>batch</td>
<td>16 nodes</td>
<td>2.66 - 12</td>
<td>32-256</td>
<td>12, 20,24</td>
<td>5d primary partition, all serial &amp; parallel jobs</td>
</tr>
<tr>
<td>short</td>
<td>8 nodes</td>
<td>4 - 12</td>
<td>48-256</td>
<td>12, 20,24</td>
<td>4h short serial &amp; parallel jobs, +96 dedicated CPU cores</td>
</tr>
<tr>
<td>huge-mem</td>
<td>1 node</td>
<td>43</td>
<td>1024</td>
<td>24</td>
<td>3d huge memory jobs, 1 node only</td>
</tr>
<tr>
<td>gpu</td>
<td>1 node, 2-8GPUs</td>
<td>2 - 10</td>
<td>24-128</td>
<td>12</td>
<td>5d GPU computing</td>
</tr>
<tr>
<td>gpushort</td>
<td>4 nodes, 2-8GPUs</td>
<td>2 - 10</td>
<td>24-128</td>
<td>12</td>
<td>4h GPU computing</td>
</tr>
<tr>
<td>interactive</td>
<td>2 nodes</td>
<td>5</td>
<td>128</td>
<td>24</td>
<td>1d for sinteractive command, longer interactive work</td>
</tr>
</tbody>
</table>

Use slurm partitions to see more details.

The debug partition and its 60 minute time limit exists for developing code and testing job scripts and simply getting used to the cluster commands. Don’t run anything here unless it is your current work focus.

Most of the time, you should not need to specify any partition (debug, interactive, or your group’s dedicated partitions). When you submit a job, there is a script (job_submit.lua) that runs in the background and automatically selects partitions. If you notice the script doing something wrong, submit an issue and we can look at it. It roughly uses this logic:
Aalto scientific computing guide

- Do you use --gres=gpu? If so, do GPU partitions.
  - Otherwise, default to batch
  - If your time limit is less than the short time limit, also add in the short partitions, too
  - If you use large amounts of memory, add hugemem.

It can be worth looking at your job’s partition list to make sure it is optimal: “slurm j $jobid”

Interactive logins

Triton mainly runs non-interactive batch jobs in its current configuration. There is a (small) partition which is meant for interactive jobs. There are two main options for running interactive shells:

- Cluster frontend (login) machine triton.aalto.fi for editing, compiling and testing code.
- Interactive jobs started with the “sinteractive” command.

We ask you to refrain from running multi-GB, many-core applications of the frontend. The login machine triton.aalto.fi is mainly intended for editing code and submission scripts, sorting your files, checking jobs and of course submitting the jobs scripts through Slurm to the actual execution nodes (called cn-something, ivy*, gpu* or tb*).

Interactive and computationally intensive applications should be run on the interactive partition. Still, to maximise the resource usage it’s best to structure your workflow such that you can use normal batch jobs.

You can access an interactive shell with the "sinteractive" command from the frontend machine triton.aalto.fi:

Launch 2 hour interactive session

```
$ sinteractive --time=2:0
```

See also the interactive usage section below for advanced examples.

Job examples

Submit a short batch job

Batch job is by default a single-CPU job that will run specified commands in series and optionally save output into a file.

A number of nodes have been dedicated for jobs that run under four hours, which makes it more likely that resources are immediately available.

Short batch example

```
#!/bin/bash
#SBATCH --time=04:00:00  # 4 hours
#SBATCH --mem=1000M      # 1G of memory

cd $WRKDIR/mydata/
srun myprog params
srun myprog2 other params
srun myprog3
```

A batch job can have as many “srun” steps as needed or just one. At the end of the day SBATCH script is just a BASH script with a set of specific directives for SBATCH. Being so, the script can be as simple as a few #SBATCH lines plus
“srun” or may consists of hundreds of BASH lines. The best practice is to join the tasks into the same job and avoid short runs (that take seconds or minutes).

**Submit an array job (batch job for repeated tasks)**

Slurm supports so-called array jobs, where one can easily handle a large number of similar jobs. An array job is essentially a set of independent jobs. In this example we run an array job consisting of 30 different array tasks. In the job script, the environment variable SLURM_ARRAY_TASK_ID is the ID of the current task. In the example below, this is used to make the application read the correct input file, and to generate output in a unique directory.

```bash
#!/bin/bash
#SBATCH --time=04:00:00
#SBATCH --mem-per-cpu=2500M
#SBATCH --array=0-29

cd $SLURM_ARRAY_TASK_ID
srun ./my_application -input input_data_$SLURM_ARRAY_TASK_ID

cd ..
```

The array indices need not be sequential. E.g. if you discover that after the above array job is finished, the job task id's 7 and 19 failed, you can relaunch just those jobs with `--array=7,19`. While the array job above is a set of serial jobs, parallel array jobs are possible. For more information, see the Slurm job array documentation.

**Submit a multithreaded job**

Programs using multiple threads must have their behaviour described to Slurm in terms of the number of threads needed. To launch a multithreaded job we tell slurm that we want a single task, but that that one task requires several CPU’s. This is done with the `--cpus-per-task=N` (or the short form `-c N`) option and should match the number of computational threads used by your application.

When moving a program from a Linux workstation to the cluster, please note than simply increasing the Slurm reservation size usually does not affect the running behavior of the program. Take a moment to see how many threads were using CPU on a workstation, and use that as a starting point (try the `top` command and press the H key to see separate threads). Not all tasks scale well to 12 (or 20, 24) threads, so run a few benchmarks in the play partition (`-p debug`) to test things before committing a lot of cluster resources to an application that may not utilize all of it. Amount of threads should be no more than number of CPU cores on the node.

For OpenMP programs the information about Slurm reservation size is passed with environment variable OMP_NUM_THREADS, which controls how many OpenMP threads will be used for the job (equal to `-n #`). However by default all allocated threads are used, so you need to specify OMP_NUM_THREADS only if you want to launch a job step using fewer than the allocated CPU’s. Other multi-threaded programs may have similar ways to control the number of threads launched. When using OpenMP, additionally one should bind threads to CPU’s with the OMP_PROC_BIND environment variable.

OpenMP example

```bash
#!/bin/bash
#SBATCH --cpus-per-task=12
#SBATCH --time=40:00
#SBATCH --mem-per-cpu=2000M
export OMP_PROC_BIND=true
srun /path/to/openMP_executable
```
Submit a MPI job

Slurm’s “srun” works as a wrapper to traditional “mpirun” command, it takes care of setting up a correct environment for MPI. For more information, see the slurm MPI guide.

Triton has several generations of different architectures, as of Oct 2018 we have Westmere, IvyBridge, Haswell, and Broadwell Xeons. They have different number on CPU cores per node: 12 for Westmere, 20 on IvyBridge, 24 on Haswell, and 28 on Broadwell.

Submit a small MPI job

A job that fit to one node: single-node job. Here we use the “-N 1” option which tells slurm to allocate all tasks on a single node. The “-n X” tells to SLURM how many MPI tasks you want to run.

Small MPI example using mvapich2

```bash
#!/bin/bash
#SBATCH --nodes=1 # on one node
#SBATCH --ntasks=4 # 4 processes
#SBATCH --time=4:00:00 # 4 hours
#SBATCH --mem-per-cpu=2000M # 2GB per process
module load gmvolf/triton-2016a
# MVAPICH + GCC + math libs modules
srun /path/to/mpi_program params
```

For “-n” less or equal to 12 this job will fit on any of the available nodes, if you put something more that 12 but below 20, it will go to either Haswell or IvyBridge nodes, and in case of up to 24 to Haswell only. Independently on the requested --n X one can always define the --constraint= and explicitly request specific CPU arch. See large MPI jobs examples below.

Submit a large MPI job

Large MPI-job, the one that does not fit to a single node. You should ask for a number of tasks that is a multiple of number of CPU cores on the node. Use the “exclusive” option to ensure that entire nodes are allocated, removing interference from other jobs and minimizing the number of nodes required to fulfill the allocation. One must specify type of requested CPU, number of tasks and corresponding number of nodes in the SBATCH script.

MPI example using Open MPI

```bash
#!/bin/bash
#SBATCH --time=2:00:00 # two hours job
#SBATCH --mem-per-cpu=1500M # 1.5GB of memory per process
#SBATCH --exclusive # allocate whole node
#SBATCH --constraint=hsw # require Haswell CPUs with 24 cores per node
#SBATCH --nodes=2 # on two nodes
#SBATCH --ntasks=48 # 48 processes to run (2 x 24 = 48)
module load goolf/triton-2016a # OpenMPI + GCC + math libs modules
srun /path/to/mpi/program params
```

-------------------- 12 CPU cores case ------------------------------

```bash
#SBATCH --constraint=wsm # require Westemers, 12 cores per node
```

(continues on next page)
Submit a hybrid MPI/OpenMP job

Batch file for running an application using a hybrid parallelization scheme with both MPI and OpenMP. Each MPI rank (process) runs a number of OpenMP threads. From the slurm perspective, this is essentially a combination of the above examples of parallel and multithreaded jobs. In this example we launch 8 MPI processes, and each MPI process runs 6 threads. The job thus uses a total of 8*6=48 cores. We explicitly require Haswell CPUs to run 4 MPI processes per node. You need to experiment with your application to see what is the best combination. Example below uses goolf-2016a with OpenMPI and GCC.

Hybrid MPI/OpenMP example using Open MPI

```bash
#!/bin/bash
#SBATCH --time=30:00
#SBATCH --mem-per-cpu=2500M
#SBATCH --exclusive
#SBATCH --constraint=hsw  # Haswells only
#SBATCH --ntasks=8        # -n, number of MPI tasks
#SBATCH --cpus-per-task=6 # -c, number of threads per MPI task
#SBATCH --nodes=2         # -N, amount of nodes
module load goolf/triton-2016a  # here we use OpenMPI as an example
export OMP_PROC_BIND=true
srun /path/to/MPI_OpenMP_program params
```

For mvapich2 you need to disable affinity, as mvapich2 has no way of specifying that each MPI rank needs N processors. One also cannot use the OpenMP affinity features, as the lack of any MPI affinity otherwise causes all MPI ranks on a node to be bound to the same cores. Example script below:

Hybrid MPI/OpenMP example using mvapich2

```bash
#!/bin/bash
#SBATCH --time=30:00
#SBATCH --mem-per-cpu=2500M
#SBATCH --exclusive
#SBATCH --constraint=[opt|wsm]
#SBATCH --ntasks=8
#SBATCH --cpus-per-task=6
#SBATCH --nodes=4
module load gmvolf/triton-2016a
export MV2_ENABLE_AFFINITY=0
srun /path/to/MPI_OpenMP_program params
```

If using other MPI flavors, please check the manual and do some tests to verify that CPU binding works correctly.
Submit a parallel (non-MPI) job

It is possible to launch parallel jobs that do not use MPI. However in this case you are responsible for setting up any necessary communication between the different tasks (processes) in the job. Depending on the job script, resources may be allocated on several nodes, so your application must be prepared to communicate over the network. The slurm “srun” command can be used to launch a number of identical executables, one for each task. Example:

```bash
#!/bin/bash
#SBATCH --time=00:01:00
#SBATCH --mem-per-cpu=500M
#SBATCH --exclusive
#SBATCH --nodes=4
srun hostname
```

This will print out the 4 allocated hostnames. The “—nodes=4” ensures that we run a task on all 4 allocated nodes. If we instead want to launch one process per allocated CPU, we can instead do “srun –ntasks=48 executable” (4*12=48).

In a case where the program in question uses Master-Worker-paradigm, where there exists a single worker that coordinates the rest, see Compute node local drives.

Most of the compute nodes are equipped with one SATA disk drive though there are some with 2 and 4. See slurm features for the full list. A node with a specific amount of drives can be requested as:

```bash
#SBATCH --gres=spindle:4
```

**GPU cards with --gres=**

See details at Slurm commands.

**Links and other additional materials**

- SLURM: Simple Linux Utility for Resource Management

**GPU Computing**

See also:

Introductory tutorial: *GPU computing* (read this first)
Overview

Triton has GPU cards from four different NVIDIA generations, as described below.

Hardware breakdown

<table>
<thead>
<tr>
<th>Card</th>
<th>total amount</th>
<th>nodes architecture</th>
<th>compute threads per GPU</th>
<th>memory per card</th>
<th>CUDA compute capability</th>
<th>Slurm feature name</th>
<th>Slurm gres name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tesla K80*</td>
<td>12</td>
<td>gpu[20-22]</td>
<td>Kepler</td>
<td>2x2496</td>
<td>2x12GB</td>
<td>kepler</td>
<td>teslak80</td>
</tr>
<tr>
<td>Tesla P100</td>
<td>20</td>
<td>gpu[23-27]</td>
<td>Pascal</td>
<td>3854</td>
<td>16GB</td>
<td>pascal</td>
<td>teslap100</td>
</tr>
<tr>
<td>Tesla V100</td>
<td>40</td>
<td>gpu[1-10]</td>
<td>Volta</td>
<td>5120</td>
<td>32GB</td>
<td>volta</td>
<td>v100</td>
</tr>
<tr>
<td>Tesla V100</td>
<td>40</td>
<td>gpu[28-37]</td>
<td>Volta</td>
<td>5120</td>
<td>32GB</td>
<td>volta</td>
<td>v100</td>
</tr>
<tr>
<td>Tesla V100</td>
<td>16</td>
<td>dgx[01-02]</td>
<td>Volta</td>
<td>5120</td>
<td>16GB</td>
<td>volta</td>
<td>v100</td>
</tr>
</tbody>
</table>

- Note: Tesla K80 cards are in essence two GK210 GPUs on a single chip
- Note: V100 cards are part of DGX machines, which were purchased by several groups and are currently special access only. They are also a different operating system, please see Nvidia DGX machines.


Using GPU nodes

GPU partitions

There are two queues governing these nodes: gpu and gpushort, where the latter is for jobs up to 4 hours. Partitions are automatically selected.

The latest details can always be found with the following command:

```
$ slurm p | grep gpu
```

GPU node allocation

For gpu resource allocation one has to request a GPU resource, with --gres=gpu:N, where N stands for number of requested GPU cards. To request a GPU with a certain architecture, use --constraint=GENERATION. To request a specific card, one must use syntax --gres=gpu:CARD_TYPE:N. See the table below for “slurm feature name” or “Slurm gres name”. For the full current list of configured SLURM gpu cards names run slurm features.

Example usages:

```
--gres=gpu:2
--gres=gpu:1 --constraint=pascal
--gres=gpu:teslap100:1
```
When using multiple GPU’s please verify that the code actually uses them with instructions given in *Monitoring GPU usage*.

### GPU nodes environment and CUDA

User environment on `gpu*` nodes is the same as on other nodes, the only difference is that they have nvidia kernel modules for Tesla cards. *CUDA* comes through `module`.

```
$ module avail cuda       # list installed CUDA modules
$ module load cuda/10.0.130 # load CUDA environment that you need
$ nvcc --version          # see actual CUDA version that you got
```

### Running a GPU job in serial

Quick interactive run:

```
$ module load cuda
$ srun --time=00:30:00 --gres=gpu:1 $WRKDIR/my_gpu_binary
```

Allocating a gpu node for longer interactive session, this will give you a shell sessions:

```
$ module load cuda
$ sinteractive --time=4:00:00 --gres=gpu:1
gpuXX$ .... run something
gpuXX$ exit
```

Run a batch job

```
$ sbatch gpu_job.sh
```

Where `gpu_job.sh` is

```
#!/bin/bash
#SBATCH --time=01:15:00 ## wallclock time hh:mm:ss
#SBATCH --gres=gpu:teslak80:1 ## one K80 requested
module load cuda
## run my GPU accelerated executable, note the --gres
srun --gres=gpu:1 $WRKDIR/my_gpu_binary
```

### Monitoring GPU usage

Currently there isn’t a good way of monitoring the gpu usage non-interactively. Interactively one can (when the job is running) `ssh` to the gpu node in question and run

```
login2$ ssh gpuxx
gpuxx$ watch --n 1 nvidia-smi
```

CTRL + C quits the command.

This shows the gpu usage with 1 second interval. The GPU utilized by process with PID X is shown in the first column of the second table. The first table lists the GPUs by their ID Checking the *Volatile GPU-Util* column gives the...
utilization of GPU. If your code uses less than 50% of the GPU you should try to improve the data loading / CPU part of your code as the GPU is underutilized.

If you run multi-GPU job you should verify that all GPUs are properly utilized. For many applications one needs to use multiple CPUs to fill the GPUs with data. With badly implemented data handling multi-GPU setups can be slower than single-GPU setups.

Development

Compiling

In case you either want to compile a CUDA code or a code with GPU support, you must do it on one of the gpu nodes (because of nvidia libs installed on those nodes only).

```bash
$ sinteractive --time=1:00:00 --gres=gpu:1   # open a session on a gpu node
$ module load cuda                             # set CUDA environment
$ nvcc cuda_code.cu -o cuda_code              # compile your CUDA code
.. or compile normally any other code with 'make'
```

Debugging

CUDA SDK provides an extension to the well-known gnu debugger gdb. Using cuda-gdb it is possible to debug the device code natively on the GPU. In order to use the cuda-gdb, one has to compile the program with option pair \(-g\) \(-G\), like follows:

```bash
$ nvcc -g -G cuda_code.cu -o cuda_code
```

See CUDA-GDB User Guide for more information on cuda-gdb.

Applications and known issues

Check the Applications for most software.

nvidia-smi utility

Could be useful for debugging, in case one wants to see the actual GPU cards available on the node. If this command returns an error, you should report that something is wrong on the node.

```bash
gpuxx$ nvidia-smi -L   # gives a list of GPU cards on the node
```

cuDNN

cudnn is available as a module. The latest version can be found with module spider cudnn. Note that (at least the later versions of) cudnn require newer cards and cannot be used on the old fermi cards. E.g. tensorflow does not run on the older fermi cards for this reason.
Nvidia MPS

Nvidia Multi-Process Service (MPS) provides a way to share a single GPU among multiple processes. It can be used to increase the GPU utilization by timesharing the GPU access, e.g. one process can upload data to the GPU while another is running a kernel. To use it one must first start the MPS server, and then CUDA calls are automatically routed via the MPS server. At the end of the job one must remember to shut it down. Example job script:

```bash
#!/bin/bash -l
#SBATCH --time=01:15:00 ## wallclock time hh:mm:ss
#SBATCH --gres=gpu:teslak80:1 ## one K80 requested

module load cuda

## Start the MPS server
CUDA_MPS_LOG_DIRECTORY=nvidia-mps srun --gres=gpu:1 nvidia-cuda-mps-control -d&

## run my GPU accelerated executable
srun --gres=gpu:1 $WRKDIR/my_gpu_binary

## Shut down the MPS server
echo "quit" | nvidia-cuda-mps-control
```

CUDA samples

There are CUDA code samples provided by Nvidia that can be useful for a sake of testing or getting familiar with CUDA. Placed at `~/CUDA_ROOT/samples`. To play with:

```bash
$ sinteractive --time=1:00:00 --gres=gpu:1
$ module load cuda
$ cp -r ~CUDA_ROOT/samples ~/WRKDIR
$ cd ~/WRKDIR/samples
$ make TARGET_ARCH=x86_64
$ ./bin/x86_64/linux/release/deviceQuery
... 
$ ./bin/x86_64/linux/release/bandwidthTest
... 
```
Attachments and useful links

- CUDA C Programming Guide
- CUDA Zone on NVIDIA
- CUDA FAQ

Grid computing

Note: not directly related to local usage covered by other pages. The whole thing is about an upper level cluster resources usage through Grid interface.

Grid computing on FGCI

The FGCI (Finnish Grid and Cloud Infrastructure) is a joint pool of resources that consists of a number of Linux clusters placed all around Finland. Triton is one of them. Being a part of FGCI Triton provides its CPU and disk space resources to grid users. Having your local account on Triton means having access to Triton local resources only, in order to use whole grid one has to get grid certificate.

To get started with grid computing on FGCI, please consult the https://research.csc.fi/fgci-user-guide pages.

Getting help

CSC provides grid users with support at grid-support(at)csc.fi.

Triton local usage vs. Grid computing

In other words should I start thinking of running on Grid? Why yes.

FGCI is homogeneous with respect to hardware/software. You may expect same Xeon CPUs connected through Infiniband and same Linux environment on all the clusters. Thus tested binaries on Triton will work on any other FGCI resource.

Grid is the best suited for from 30 minutes up to 24 hours long jobs. It is for production runs mainly, you do the development/testing locally on Triton and then send your jobs to the grid.

One can run MPI or large number of serial jobs, no CPU units/limits, no overloaded queues, there always be a free slot for your job on one of FGCI clusters.

Grid job management can be done from anywhere you prefer, including Triton, CSC servers or your department workstation. Grid certificate is independent on your Aalto / Triton / CSC account and valid for one year.

Grid computing 2

FGI pages at CSC

Sneak peek: http://www.nordugrid.org/monitor/
Obtaining personal certificate

https://confluence.csc.fi/display/fgi/Getting+started+with+grid

Installing client tools

http://www.nordugrid.org/documents/arc-client-install.html

Using the grid

https://confluence.csc.fi/display/fgi/FGI+Hands+on+tutorial+material

Monitoring jobs

Basics

Before you start, if you have SLURM experience but new to Triton, you may want to check out the `slurm` command on Triton’ paragraph below. There we introduce Triton specific tool that is widely used for jobs monitoring and many other issues.

There are two quick ways to see your own jobs. One is to list every job and their current priority in the queues.

```
$ slurm q
PRIORITY JOBID PARTITION NAME ST TIME START_TIME
--NODELIST(REASON)
12971826 batch-wsm DA_vk_400k_5M 0:00 N/A PENDING
    (Priority)
12971825 batch-wsm DA_vk_400k_50M 0:00 N/A PENDING
    (Priority)
12971818 batch-hsw DA_vk_400k_5M 0:06 2016-10-17T15:12 RUNNING
    ivb28
12971817 short-ivb update.12.sh 0:06 2016-10-17T15:12 RUNNING
    ivb13
12971816 batch-ivb FI_vk_2000k_50 0:07 2016-10-17T15:12 RUNNING
    ivb28
...
```

But mostly a simple overview will do. Here is the same information as above, only this time formatted for brevity. The leftmost value is the combined job count.

```
$ slurm qq
1 PARTITION CPUS NODES MIN_MEM FEATURES QOS STATE REASON
1 batch-wsm 1 1 4G ivb|wsm| normal PENDING AssocGrpMemRunMinutes
1 batch-wsm 1 1 8G ivb|wsm| normal PENDING AssocGrpMemRunMinutes
242 batch-wsm 1 1 25G (null) normal PENDING AssocGrpMemRunMinutes
2 coin 1 1 4G (null) normal RUNNING None
15 batch-wsm 1 1 4G (null) normal RUNNING None
2 coin 1 1 4G (null) normal RUNNING None
...
```

For “watching” your jobs progress, use ‘watch’ option

```
slurm watch q
```
Since the cluster is in heavy use most of the time, there are other users whose jobs will of course affect what happens with yours. The state of the entire queue, including running either `slurm short` or `slurm ss`.

```
slurm s
```

With the `s` or `short` option, the output is sorted by priority and reflects the scheduler’s execution order as nodes become available. The queue position can change at any time, either from new or submissions or based on historical usage accounting.

You can choose to display the one partition you are interested in:

```
slurm ss gpu
```

Show detailed information about running job(s):

```
slurm j
```

### Job status while pending

There are a number of reasons that your job may be sitting in the queue. Listing your pending jobs with `squeue -u $USER -t PD` will help determine why your job is not running. Look at the `NODELIST(REASON)`. A pending job may have these reasons:

- **(Priority):** Other jobs have priority over your job. Just wait.
- **(Resources):** Your job has enough priority to run, but there aren’t enough free resources to run it. Just wait.
- **(ReqNodeNotAvail):** You request something that is not available. Check memory requirements per CPU, CPUs per node. Possibly time limit is the issue. Could be that due to scheduled maintenance break, all nodes are reserved and thus your `-t` parameter can’t be larger than time left till the break.
- **(QOSResourceLimit):** Your job exceeds the QOS limits. The QOS limits include wall time, number of jobs a user can have running at once, number of nodes a user can use at once, etc. This may or may no be a permanent status. If your job requests a wall time greater than what is allowed or exceeds the limit on the number of nodes a single job can use, this status will be permanent. However, your job may be in this status if you currently have jobs running and the total number of jobs running or aggregate node usage is at your limits. In this case, jobs in this state will become eligible when your existing jobs finish.
- **(AssociationResourceLimit):** The job exceeds some limit set on the association. On triton, this in practice means the per-user GrpCPURunMins limit, which currently is 1.5M minutes per user. Wait a while for running jobs to proceed, so that new jobs may start. Also, shorter job time limits help. See GrpCPURunMins visualizer.

In case of the first two one can check currently estimated time the job will be started. Run `slurm j <jobid>`, look at `StartTime=`

### Job states

Possible states for jobs are:

- **PENDING (PD)**
- **RUNNING (R)**
- **SUSPENDED (S)**
- **COMPLETING (CG)**
Aalto scientific computing guide

• COMPLETED (CD)
• CONFIGURING (CF)
• CANCELLED (CA)
• FAILED (F)
• TIMEOUT (TO)
• PREEMPTED (PR)
• NODE_FAIL (NF)``.

Modifying the job after submission

The question asked time to time: “Can one modify job parameters after it has been submitted?”. The answer is, yes it is possible, but only some parameters. For instance change memory/CPU requirements for pending job or set another time limit off running/pending job. Think carefully before you submit a job, but if you ended up in the situation that modification is needed, please contact your Triton support team member.

Needless to say that there is no way to impact on your job priority and make sure that it goes to run asap?

Viewing finished jobs

Information about finished and cancelled jobs are available via the slurm history command. Most notable pieces are memory use and also exit code, in case the jobs did not finish cleanly.

```
$ slurm history 2hours
JobID JobName Start MaxVMem MaxRes TotalCPU Elapsed  Tasks CPUS Nodes ExitCode State
-1Tasks CPUs Nodes ExitCode State
1052748 helloworld 2012-04-10T19:05 00:00.015 00:00:00 - none 1 1 1:0 FAILED
1052751 testarr 2012-04-10T19:07 00:00.074 00:02:30 - none 5 1 0:0 COMPLETED
1052751.0 runtask 2012-04-10T19:08 99M 1M 00:00.002 00:00:30 - 1 1 1 0:0 COMPLETED
1052751.1 runtask 2012-04-10T19:08 99M 1M 00:00.003 00:00:30 - 1 1 1 0:0 COMPLETED
1052751.2 runtask 2012-04-10T19:08 99M 1M 00:00.003 00:00:30 - 1 1 1 0:0 COMPLETED
1052751.3 runtask 2012-04-10T19:08 99M 1M 00:00.003 00:00:30 - 1 1 1 0:0 COMPLETED
1052751.4 runtask 2012-04-10T19:08 99M 1M 00:00.003 00:00:30 - 1 1 1 0:0 COMPLETED
```

Output time information is displayed as days-hours:minutes:seconds.

Recognized time forms for the input parameter are n min, n hours, n days, n weeks (without space).

*Elapsed* is the wall clock time from job start to finish.

*MaxVMem* is the highest amount of virtual memory your program allocated during its lifetime. If the slurm job’s memory limit is set below it, your job would be killed.
MaxRes is the resident (physical) memory the program actually used of its virtual memory allocation, which may be of interest when monitoring program behavior.

### Cancelling jobs

```bash
$ scancel  # cancel a job
$ scancel `echo 5205484..5205533` # cancel jobs in the range
$ scancel --state=PENDING --user=$USER --partition=debug # kill all of your pending jobs on debug queue
```

### Job priority

Triton queues are not first-in first-out, but “fairshare”. This means that every person has a priority. The more you run the lower your user priority. As time passes, your user priority increases again. The longer a job waits in the queue, the higher its job priority goes. So, in the long run (if everyone is submitting an never-ending stream of jobs), everyone will get exactly their share.

Once there are priorities, then: jobs are scheduled in order of priority, then any gaps are backfilled with any smaller jobs that can fit in. So small jobs usually get scheduled fast regardless.

**Warning:** from this point on, we get more and more technical, if you really want to know the details. Summary at the end.

What’s a share? Currently shares are based on department and their respective funding of Triton (sshare). Shares are shared among everyone in the department, but each person has their own priority. Thus, for medium users, the 2-week usage of the rest of your department can affect how fast your jobs run. However, again, things are balanced per-user within departments. (However, one heavy user in a department can affect all others in that department a bit too much, we are working on this)

Your priority goes down via the “job billing”: roughly time×power. CPUs are billed at 1/s (but older, less powerful CPUs cost less!). Memory costs .2/GB/s. But: you only get billed for the max of memory or CPU. So if you use one CPU and all the memory (so that no one else can run on it), you get billed for all memory but no CPU. Same for all CPUs and little memory. This encourages balanced use. (this also applies to GPUs).

GPUs also have a billing weight, currently tens of times higher than a CPU billing weight for the newest GPUs. (In general all of these can change, for the latest info see search `BillingWeights` in `/etc/slurm/slurm.conf`).

If you submit a long job but it ends early, you are only billed for the actual time you use (but the longer job might take longer to start at the beginning). Memory is always billed for the full reservation even if you use less, since it isn’t shared.

The “user priority” is actually just a record how much you have consumed lately (the billing numbers above). This number goes down with a half-life decay of 2 weeks. Your personal priority your share compared to that, so we get the effect described above: the more you (or your department) runs lately, the lower your priority.

If you want your stuff to run faster, the best way is to more accurately specify your time (may make that job can find a place sooner) and memory (avoids needlessly wasting your priority).

While your job is pending in the queue SLURM checks those metrics regularly and recalculates job priority constantly. If you are interested in details, take a look at multifactor priority plugin page (general info) and depth-oblivious fair-share factor for what we use specifically (warning: very in depth page). On Triton, you can always see the latest billing weights in `/etc/slurm/slurm.conf`.

Numerically, job priorities range from 0 to $2^{32}-1$. Higher is sooner to run, but really the number doesn’t mean much itself.

These commands can show you information about your user and job priorities:

#### 6.1. Triton cluster
slurm\ s \ list\ of\ jobs\ per\ user\ with\ their\ current\ priorities
slurm\ full \ as\ above\ but\ almost\ all\ of\ the\ job\ parameters\ are\ listed
slurm\ shares \ displays\ usage\ (RawUsage)\ and\ current\ FairShare\ weights\ (FairShare,\ higher\ is\ better)\ values\ for\ all\ users
slurm\ j \ <jobid> \ shows\ <jobid>\ detailed\ info\ including\ priority,\ requested\ nodes\ etc.

\textit{tl;dr:} Just select the resources you think you need, and slurm tries to balance things out so everyone gets their share. The best way to maintain high priority is to use resources efficiently so you don’t need to over-request.

**slurm command on Triton**

A nice tool originally developed by Tapio Leipälä specifically for Triton user needs and developed by Triton support team nowadays.

The slurm command can show most often needed information about jobs, resources and the cluster state. It’s a wrapper script to the native SLURM commands. New features are added from time to time. Running it without parameters prints a list of available commands. Most have some soft of shortcuts for convenience.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slurm q;slurm qq</td>
<td>Status of your queued jobs (long/short)</td>
</tr>
<tr>
<td>slurm partitions</td>
<td>Overview of partitions (A/I/O/T=active,idle,other,total)</td>
</tr>
<tr>
<td>slurm cpus &lt;partition&gt;</td>
<td>list free CPUs in a partition</td>
</tr>
<tr>
<td>slurm history[1day,2hour,...]</td>
<td>Show status of recent jobs</td>
</tr>
<tr>
<td>seff &lt;jobid&gt;</td>
<td>Show percent of mem/CPU used in job</td>
</tr>
<tr>
<td>slurm j &lt;jobid&gt;</td>
<td>Job details (only while running)</td>
</tr>
<tr>
<td>slurm s;slurm ss &lt;partition&gt;</td>
<td>Show status of all jobs</td>
</tr>
<tr>
<td>sacct</td>
<td>Full history information (advanced, needs args)</td>
</tr>
</tbody>
</table>

**Full slurm command help:**

```
$ slurm

Show or watch job queue:
  slurm [watch] queue  show own jobs
  slurm [watch] q  show user's jobs
  slurm [watch] quick  show quick overview of own jobs
  slurm [watch] shorter  sort and compact entire queue by job size
  slurm [watch] short  sort and compact entire queue by priority
  slurm [watch] full  show everything
  slurm [w] [q|qq|ss|sf]  shorthands for above!
  slurm qos  show job service classes
  slurm top [queue|all]  show summary of active users

Show detailed information about jobs:
  slurm prio [all|short]  show priority components
  slurm jjob  show everything else
  slurm steps  show memory usage of running srun job steps

Show usage and fair-share values from accounting database:
  slurm h|history  show jobs finished since, e.g. "1day" (default)
  slurm shares

Show nodes and resources in the cluster:
  slurm p|partitions  all partitions
  slurm n|nodes  all cluster nodes
```

(continues on next page)
slurm c|cpus total cpu cores in use
slurm cpus cores available to partition, allocated and free
slurm cpus jobs cores/memory reserved by running jobs
slurm cpus queue cores/memory required by pending jobs
slurm features List features and GRES

Examples:
slurm q
slurm watch shorter
slurm cpus batch
slurm history 3hours

Other advanced commands (many require lots of parameters to be useful):

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>squeue</td>
<td>Full info on queues</td>
</tr>
<tr>
<td>sinfo</td>
<td>Advanced info on partitions</td>
</tr>
<tr>
<td>slurm nodes</td>
<td>List all nodes</td>
</tr>
</tbody>
</table>

Native slurm commands

While Triton has a slurm utility that hides most of original SLURM commands, you still may want to learn more. If need something else that slurm can not do, the native commands with their full functionality are at your service. For the details, please consult corresponding man pages (man squeue, etc).

- **squeue** – view information about jobs located in the Slurm scheduling queue

  $ squeue -n gpu[1-22]  # reports only jobs allocated to specific nodes
  $ squeue -t PD -i 5 -u $USER  # reports your pending jobs, with the 5s interval

- **sinfo** – view node & partition information
- **sshare** – show statistics from the accounting database
- **scontrol** – various function, end user mostly interested in scontrol show...

  scontrol show node ivb1  # show specific node config

- **sprio** - Show calculated priority factors for jobs waiting in the queue
- **sacct** - Historical info about jobs

Customizable output for slurm

The slurm command output can be customized. Look in the for variable names in /usr/local/bin/slurm and place them into your own $HOME/.config/slurmvars file.

For example, more detailed node info for those interested to know which kind of machines are free. This changes the look of slurm partitions.

fmt_s_parts="%10P %.10l %.15F %8f %N"
Monitoring jobs’ usage of the filesystem

Our Lustre servers collect so-called jobstats - statistics of Lustre usage by individual Slurm jobs. These statistics can be viewed by a tool called jobstats. To view jobstats for your job, you need to do the following:

```
module load jobstats
jobstats <jobid> <jobid> ...
```

Do note that the job statistics are collected from our storage servers every 10 seconds, but are only transferred to our statistics servers every 5 minutes. Thus smalls that take less than 5 minutes might show incorrect statistics.

Please do not run too many queries too often, as we’re still testing the capabilities of our statistics server.

Output of the tool looks something like this:

```
54324759
Bytes read (% of load on a single server) -1
Bytes read (Max) 0
Bytes read (Total) 0
Bytes written (% of load on a single server) 88.88888888888889
Bytes written (Max) 6442450944
Bytes written (Total) 724757312
Files closed (Total) -1
Files opened (Total) -1
Getattr calls made (Total) 0
Read calls made (Total) 0
Setattr calls made (Total) 0
Write calls made (Total) 1728
```

Currently the jobstats history only goes back in time for couple of weeks. This is due to the amount of statistics that our monitoring records.

How to read jobstats

Any values with -1 mean that either that there wasn’t any calls that were captured or that the percentage ratio could not be calculated.

The different lines of the jobstats output are explained below:

- **Bytes read (% of load on a single server):** This statistic shows the percentage of bytes that were read from a single Lustre server. If this is close to a 100%, see the recommendations below for avoiding hotspots.

- **Bytes read (Max):** This statistic shows the number of bytes that were read from a single Lustre server. This number divided by the total bytes read will produce the percentage of load on a single Lustre server.

- **Bytes read (Total):** This statistic shows the total number of bytes read by your job. If the number is very high, see the recommendations below for minimizing your data loading.

- **Bytes written (% of load on a single server):** Similar to bytes read percentage, but for bytes written.

- **Bytes written (Max):** Similar to bytes read maximum, but for bytes written.

- **Bytes written (Total):** Similar to bytes read total, but for bytes written.

- **Files closed (Total):** Total number of files closed during the job runtime. If this is very high, please check below for recommendations on minimizing the number of files opened/closed.

- **Files opened (Total):** Similar to files closed, but for files opened.
• Getattr calls made (Total): Total number of getattr metadata calls made during job runtime. If this is high, please check below for recommendations on minimizing the number of metadata calls.

• Read calls made (Total): Number of file read calls made during the job runtime (byte reads). If high, please check below for recommendations on how to minimize the number of read calls.

• Setattr calls made (Total): Similar to getattr calls, but this time for setattr calls.

• Write calls made (Total): Similar to read calls, but this time for write calls.

My job shows a close to 100% bytes read/write load on a single server

If your job has a high percentage of load on a single server AND the number of bytes read/written is high, you might cause hotspots on our storage servers.

As each file is by default striped on a single server, having a large file read/written by your code will put the load of reading/writing the file on a single server.

Of course, if you read/write a large amount of data you should first look at the instructions on that as well so that the overall amount of data that’s being read/written will be minimized.

Besides that, you might want to split your data to smaller pieces (1-10GB, depending on the size of the original data) or to look at our Lustre-page for instructions on how to stripe your large files to multiple storage servers. Both methods distribute the load of reading/writing to multiple servers.

However, you should not stripe small files as that will increase the filesystem load. If uncertain, please ask us admins about helping you with a good IO workflow.

My job shows a very high read/write byte count

If your job reads/writes a lot of data the main question you should ask is whether all of the reads/writes are necessary. For example, you might ask:

1. Do I use/need all of the output my code is generating?
2. If I create checkpoints of the state of my program execution, is the rate of checkpointing set to be too fast?
3. Could I do my IO against the local disks or ramdisk and only once copy my input there/copy my output out?
4. Is some of this load created by temporary files? Could I divert some of this load to local disks?
5. Am I constantly reading the same files again and again? Could I place the files to the local disks/ramdisk during the job runtime?
6. Do I use efficient binary output format or do I save data in an ASCII textfile which requires more space?
7. Am I loading a dataset, but only using a piece of it during job execution? Could I split the dataset to pieces before I run my code?

If you have any questions or do not know the answer how your program does its IO, do not hesitate to ask us admins on helping you figure out a good IO workflow.
My job opens/closes a high number of files

If your job opens/closes a large number of files it might either:

1. Constantly open/close the input/output files. This is usually unnecessary and can be fixed by keeping the input/output files open during job runtime or by reducing the rate of output.

2. Open a large number of input/output files. If you’re working with a large number of small files you might want to look at our pages on how to use local disks and tar to group up your small files into larger collections.

3. Your code might open a lot of libraries stored in the Lustre filesystem. Are you compiling code? Use local disks or ramdisk for that. Are you loading an anaconda environment or R libraries from the Lustre filesystem? Move them to your $HOME-directory.

My job makes a high number of getattr/setattr-calls

Do you run something like find ..., ls -l ... or chmod ... in your scripts? Commands like these do getattr/setattr queries towards our metadata servers. Are these commands necessary for running your program?

These kinds of calls also happen if you untar/unzip files to the Lustre filesystem.

My job makes a high number of read/write-calls

A high number of read/write calls can happen for multitude of reasons.

Firstly, as number of bytes read/written = size of a block read/written* number of read/write calls, a large number of read/write calls might indicate that your code is read/write its input/output in small chunks. Using ASCII data, small files or read/print statements in your code can easily create a huge number of read/write calls. For example, a print statement is typically written into a buffer of size 4-64KB. Thus writing a 10MB file using print statements can create up to thousands of write calls. Using binary formats will usually solve this problem as they usually write in much larger chunks (typically 1-4MB).

Secondly, a large number of read/write operations might occur if you’re doing lots of random read operations. This happens easily when using database formats such as lmdb or sqlite. Lustre is not the best format for random reads or for database interaction. You might want to look into using local disks or ramdisk for storing your database during the job runtime.

If you’re reading/writing a lot of data or opening/closing a lot files, this number is usually very high as well. Thus it is usually good to look at the instructions for those cases as well.

Libraries

BLAS

Basic Linear Algebra Subroutine (BLAS) is a de facto application programming interface standard for publishing libraries to perform basic linear algebra operations such as vector and matrix multiplication (from wikipedia).

On triton a number of different BLAS implementations are available. The recommended ones are MKL and OpenBLAS. Other available BLAS libraries are ATLAS, GotoBLAS2, ACML, and the Netlib reference BLAS. For benchmark results see

DGEMM benchmark on triton
Using MKL

In order to use the MKL library you need to load the module. MKL is provided both in the “mkl” modules and in the “intel” modules; the “intel” module additionally give you the Intel compilers and debuggers. Linking with MKL is a bit tricky and the exact link options varies from version to version. Intel provides a webpage to build the correct linking options at https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor.

Using OpenBLAS

OpenBLAS is installed on all the triton nodes on the default library directory (/usr/lib64). 3 different variants of the library are provided:

- Serial version: Link using “-lopenblas”
- OpenMP version: Link using “-lopenblaso”
- pthreads version: Link using “-lopenblasp”

Other BLAS libraries

In general MKL and OpenBLAS are recommended since they both provide good performance on all the node types we have. Other BLAS libraries have various issues such as crashing when running on the incorrect node (e.g. running an Haswell optimized library on an Westmer node or vice versa), or poor performance. In particular, the Netlib reference BLAS has VERY poor performance and should be avoided at all cost. Use it only for testing or if you need to debug numeric output. As can be seen on the DGEMM benchmark results performance for large matrices is an order of magnitude worse than the optimized versions. For a real example see for instance tracker.triton.aalto.fi/issue194 where 50% performance loss was seen for a complete application.

LAPACK

Linear Algebra Package (LAPACK) is a library of numerical linear algebra algorithms, built on top of BLAS. The recommended LAPACK implementation on triton is MKL. Alternatively, our OpenBLAS modules and libraries also contain the LAPACK library compiled against the OpenBLAS BLAS library.

Scalapack

MKL contains an implementation of ScaLAPACK as well, please try to use that one first. Again, see the BLAS section for how to link with MKL.

Benchmark is done with full Scalapack LIN/EIG testsuite with 24 processors. Scalapack is compiled with -O3 using architecture optimized gotoblas2. Given numbers are WallClockTimes in seconds.

<table>
<thead>
<tr>
<th>Library</th>
<th>Xeon processors</th>
<th>Opteron processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xeon-gcc optimized scalapack</td>
<td>342s</td>
<td>-na-</td>
</tr>
<tr>
<td>Opteron-gcc optimized scalapack</td>
<td>338s</td>
<td>291s</td>
</tr>
<tr>
<td>Intel-compiler-mkl</td>
<td>-na-</td>
<td>-na-</td>
</tr>
</tbody>
</table>

Scalapck libs are available under /share/apps/scalapack/2.0.1/
FFT

The FFTW library is available on triton, in several different variants. The recommended one is the one provided by MKL; see the BLAS section above for how to link to it.

<table>
<thead>
<tr>
<th>Library</th>
<th>module</th>
<th>Link line</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFTW 3.2.1</td>
<td>•</td>
<td>-lfftw3 / -lfftw3_threads / -lfftw3f / -lfftw3f_threads / -lfftw3l / -lfftw3l_threads</td>
</tr>
<tr>
<td>FFTW 3.3.2</td>
<td>fftw/3.3.2</td>
<td>-lfftw3 / -lfftw3_mpi</td>
</tr>
<tr>
<td>MKL</td>
<td>intel/VERSION or mkl/VERSION</td>
<td>See <a href="https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor">https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor</a></td>
</tr>
</tbody>
</table>

Links

- BLAS at www.netlib.org/blas
- LAPACK at www.netlib.org/lapack
- GotoBLAS FAQ at [www.tacc.utexas.edu/tacc-projects/gotoblas2/faq](http://www.tacc.utexas.edu/tacc-projects/gotoblas2/faq)
- ACML User’s Guide (pdf file) at developer.amd.com/assets

Storage: local drives

See also:

*the storage tutorial.*

Local disks on computing nodes are the preferred place for doing your IO. The general idea is use network storage as a backend and local disk for actual data processing.

- In the beginning of the job cd to /tmp and make a unique directory for your run
- copy needed input from WRKDIR to there
- run your calculation normally forwarding all the output to /tmp
- in the end copy relevant output to WRKDIR for analysis and further usage

Pros

- You get better and steadier IO performance. WRKDIR is shared over all users making per-user performance actually rather poor.
- You save performance for WRKDIR to those who cannot use local disks.
- You get much better performance when using many small files (Lustre works poorly here).
- Saves your quota if your code generate lots of data but finally you need only part of it
- In general, it is an excellent choice for single-node runs (that is all job’s task run on the same node).

Cons

- Not feasible for huge files (>100GB). Use WRKDIR instead.
- Small learning curve (must copy files before and after the job).
• Not feasible for cross-node IO (MPI jobs). Use WRKDIR instead.

**How to use local drives on compute nodes**

NOT for the long-term data. Cleaned every time your job is finished.

You have to use `--gres=spindle` to ensure that you get a hard disk (note 2019-january: except GPU nodes).

/\tmp\ is a bind-mounted user specific directory. Directory is per-user (not per-job that is), if you get two jobs running on the same node, you get the same /\tmp\.

**Interactively**

How to use /\tmp\ when you login interactively

```
$ sinteractive --time=1:00:00  # request a node for one hour
(node)$ mkdir /tmp/$SLURM_JOB_ID  # create a unique directory, here we use
(node)$ cd /tmp/$SLURM_JOB_ID
... do what you wanted ...
(node)$ cp your_files $WRKDIR/my/valuable/data  # copy what you need
(node)$ cd; rm -rf /tmp/$SLURM_JOB_ID  # clean up after yourself
(node)$ exit
```

**In batch script**

Batch job example that prevents data lost in case program gets terminated (either because of `scancel` or due to time limit).

```
#!/bin/bash

#SBATCH --time=12:00:00
#SBATCH --mem-per-cpu=2500M  # time and memory requirements

mkdir /tmp/$SLURM_JOB_ID  # get a directory where you will send all output from your program
cd /tmp/$SLURM_JOB_ID

## set the trap: when killed or exits abnormally you get the output copied to $WRKDIR/$SLURM_JOB_ID anyway
trap "mkdir $WRKDIR/$SLURM_JOB_ID; mv -f /tmp/$SLURM_JOB_ID $WRKDIR/$SLURM_JOB_ID; exit" TERM EXIT

## run the program and redirect all IO to a local drive
## assuming that you have your program and input at $WRKDIR
srun $WRKDIR/my_program $WRKDIR/input > output

mv /tmp/$SLURM_JOB_ID/output $WRKDIR/SOMEDIR  # move your output fully or partially
```

6.1. Triton cluster
Batch script for thousands input/output files

If your job requires a large amount of files as input/output using tar utility can greatly reduce the load on the $WRKDIR-filesystem.

Using methods like this is recommended if you’re working with thousands of files.

Working with tar balls is done in a following fashion:

1. Determine if your input data can be collected into analysis-sized chunks that can be (if possible) re-used
2. Make a tar ball out of the input data (tar cf <tar filename>.tar <input files>)
3. At the beginning of job copy the tar ball into /tmp and untar it there (tar xf <tar filename>.tar)
4. Do the analysis here, in the local disk
5. If output is a large amount of files, tar them and copy them out. Otherwise write output to $WRKDIR

A sample code is below:

```bash
#!/bin/bash
#SBATCH --time=12:00:00 # time and memory requirements
#SBATCH --mem-per-cpu=2000M
mkdir /tmp/$SLURM_JOB_ID # get a directory where you will put your data
# copy tarred input files
cd /tmp/$SLURM_JOB_ID
trap "rm -rf /tmp/$SLURM_JOB_ID; exit" TERM EXIT # set the trap: when killed or exits abnormally you clean up your stuff
tar xf input.tar # untar the files
srun input/* # do the analysis, or what ever else
tar cf output.tar output/* # tar output
mv output.tar $WRKDIR/SOMEDIR # copy results back
```

Storage: Lustre (scratch)

See also:

*the storage tutorial.*

Lustre is scalable high performance file system created for HPC. It allows MPI-IO but mainly it provides large storage capacity and high sequential throughput for cluster applications. Currently the total capacity is 2PB. The basic idea in Lustre is to spread data in each file over multiple storage servers. With large (larger than 1GB) files Lustre will significantly boost the performance.
Working with small files

As Lustre is meant for large files, the performance with small (smaller than 10MB) files will not be optimal. If possible, try to avoid working with multiple small files.

Note: Triton has a default stripe of 1 already, so it is by default optimized for small files (but it’s still not that great). If you use large files, see below.

If small files are needed (i.e. source codes) you can tell Lustre not to spread data over all the nodes. This will help in performance.

To see the striping for any given file or directory you can use following command to check status

```
lfs getstripe -d /scratch/path/to/dir
```

You can not change the striping of an existing file, but you can change the striping of new files created in a directory, then copy the file to a new name in that directory.

```
lfs setstripe -c 1 /scratch/path/to/dir
cp somefile /scratch/path/to/dir/newfile
```

Working with lots of small files

Large datasets which consist mostly of small (<1MB) files can be slow to process because of network overhead associated with individual files. If it is your case, please consult Compute node local drives page, see the tar example over there or find some other way to compact your files together into one.

Working with large files

By default Lustre on Triton is configured to stripe a single file over a single OST. This provides the best performance for small files, serial programs, parallel programs where only one process is doing I/O, and parallel programs using a file-per-process file I/O pattern. However, when working with large files (>> 10 GB), particularly if they are accessed in parallel from multiple processes in a parallel application, it can be advantageous to stripe over several OST’s. In this case the easiest way is to create a directory for the large file(s), and set the striping parameters for any files subsequently created in that directory:

```
cd $WRKDIR
mkdir large_file
lfs setstripe -c 4 large_file
```

The above creates a directory large_file and specifies that files created inside that directory will be striped over 4 OST’s. For really really large files (hundreds of GB’s) accessed in parallel from very large MPI runs, set the stripe count to “-1” which tells the system to stripe over all the available OST’s.

To reset back to the default settings, run

```
lfs setstripe -d path/to/directory
```
Aalto scientific computing guide

Monitoring Lustre usage

You can monitor your jobs’ Lustre usage with our *jobstats-utility*.

Lustre: common recommendations

- Minimize use of `ls -l` and `ls --color` when possible

Several excellent recommendations are at

- [http://www.nics.tennessee.edu/computing-resources/file-systems/io-lustre-tips](http://www.nics.tennessee.edu/computing-resources/file-systems/io-lustre-tips)

They are fully applicable to our case.

Be aware, that being a high performance filesystem Lustre still has its own bottlenecks, and even non-proper a usage by a single user can get whole system in stuck. See the recommendations at the link above how to avoid those potential situations. Common Lustre troublemakers are `ls -lR`, creating many small files, `rm -rf`, small random i/o, heavy bulk i/o.

For advanced user, these slides can be interesting: [https://www.eofs.eu/fileadmin/lad2012/06_Daniel_Kobras_S_C_Lustre_FS_Bottleneck.pdf](https://www.eofs.eu/fileadmin/lad2012/06_Daniel_Kobras_S_C_Lustre_FS_Bottleneck.pdf)

MPI on Triton

The basic parallel programming model for PC-clusters is message passing. MPI (Message Passing Interface) is a library specification for message-passing, proposed, nowadays, as a standard. OpenMPI and MVAPICH are two different (but doing the same) MPI implementations, both are installed on triton. MVAPICH is the recommended one. MPI is suitable for both distributed memory computers (thus through interconnect between nodes) and shared memory architectures (computing within one node).

Check out the latest version of MVAPICH / OpenMPI with

```bash
$ module avail
```

By setting up your environment with `module`, you will set all the variables like `$MPIRUN`, `$MPIHOME` etc.

Load MVAPICH2:

```bash
$ module load mvapich2/1.8a2-gcc-4.4.6
```

Load OpenMPI (for software compiled on recently updated SL6.2):

```bash
$ module load openmpi/1.4.5-gcc-4.4.6
```

Load OpenMPI (for software compiled on SL6.1):

```bash
$ module load compat-openmpi-x86_64
```

Compiling after that is quite straightforward:

```bash
$ mpif90 your_code.f90 -o your_code
$ mpicc your_code.c -o your_code
```

Parallel programs are to be run as batch jobs, even test runs. See Executing jobs / Batch system for examples of running in parallel.
You have code, you want it to run fast. This is what Triton is for. But how do you know if your code is running as fast as it can? We are scientists, and if things aren’t quantified we can’t do science on them. Programming can often seem like a black box: modern computers are extremely complicated, and people can’t predict what is actually making code fast or slow anymore. Thus, you need to profile your code: get detailed performance measurements. These measurements let you know how to make it run faster.

There are many tools for profiling, and it really is one of the fundamental principles for any programming language. You really should learn how to do quick profile just to make sure things are OK, even if you aren’t trying to optimize things: you might find a quick win even if you didn’t write the code yourself (for example, 90% of your time is spent on input/output).

This page is under development, but so far serves as an introduction. We hope to expand it with specific Triton examples.

### Summary: profiling on Linux

First off, look at your language-specific profiling tools.

- Generic Linux profiling tools (big and comprehensive list, also some presentations): [http://www.brendangregg.com/linuxperf.html](http://www.brendangregg.com/linuxperf.html)


### CPU profiling

This can give you a list of where all your processor time is going, either per-function or per-line. Generally, most of your time is in a very small region of your code, and you need to know what this is in order to improve just that part.

See the C and Python profiling example above.

### GNU gprof

gprof is a profiler based on instrumenting your code (build with -pg). It has relatively high overhead, but gives exact information e.g. for the number of times a function is called.
Perf

Perf is a sampling profiler, which periodically samples events originating e.g. from the CPU performance monitoring unit (PMU). This generates a statistical profile, but the advantage is that the overhead is very low (single digit %), and one can get timings at the level of individual asm instructions. For a simple example, consider a (naive) matrix multiplication program:

Compile the program (-g provides debug symbols which will be useful later on, at no performance cost):

```
$ gfortran -Wall -g -O3 mymatmul.f90
```

Run the program via the profiler to generate profile data:

```
$ perf record ./a.out
```

Now we can look at the profile:

```
$ perf report
# Samples: 1251
#
# Overhead Command Shared Object Symbol
# .......... ................. ..................................... ...
# 85.45% a.out ./a.out [.] MAIN_
# 4.24% a.out /usr/lib/libgfortran.so.3.0.0 [.] __gfortran__arandom_r4
# 3.12% a.out /usr/lib/libgfortran.so.3.0.0 [.] kiss__random__kernel
```

So 85% of the runtime is spent in the main program (symbol MAIN_), and most of the rest is in the random number generator, which the program calls in order to generate the input matrices.

Now, let's take a closer look at the main program:

```
$ perf annotate MAIN_
------------------------------------------------
Percent \| Source code & Disassembly of a.out
------------------------------------------------
: : :
: : :
: Disassembly of section .text:
: :
: 00000000004008b0 <MAIN__>
...
: c = 0.
: :
: do j = 1, n
: do k = 1, n
: do i = 1, n
: c(i,j) = c(i,j) + a(i,k) * b(k,j)
30.12 : 400a40: 0f 28 04 01 movaps (%rcx,%rax,1),%xmm0
4.92 : 400a44: 0f 59 c1 mulps %xmm1,%xmm0
12.36 : 400a47: 0f 58 04 02 addps (%rdx,%rax,1),%xmm0
40.73 : 400a4b: 0f 29 04 02 movaps %xmm0,(%rdx,%rax,1)
9.65 : 400a4f: 48 83 c0 10 add 50x10,%rax
```

Unsurprisingly, the inner loop kernel takes up practically all the time.
For more information on using perf, see the perf tutorial at
https://perf.wiki.kernel.org/index.php/Tutorial

Input/output profiling

This will tell you how much time is spent reading and writing data, where, and what type of patterns it has (big reads, random access, etc). Note that you can see the time information when CPU profiling: if input/output functions take a lot of time, you need to improve IO.

/usr/bin/time -v prints some useful info about IO operations and statistics.

Lowest level: use strace to print the time taken in every system call that accesses files. This is not that great:

```bash
# Use strace to print the total bytes
strace -e trace=desc $command |& egrep 'write' | awk --field-separator='=' '{ x+=$NF } END { print x }'
strace -e trace=desc $command |& egrep 'read' | awk --field-separator='=' '{ x+=$NF } END { print x }'
# Number of calls only
strace -e trace=file -c $command
```

Memory profiling

Less common, but it can tell you something about what memory is being used.

If you are making your own algorithms, memory profiling becomes more important because you need to be sure that you are using the memory hierarchy efficiently. There are tools for this.

MPI and parallel profiling

mpiP

mpiP: Lightweight, Scalable MPI Profiling http://mpip.sourceforge.net/. Collects statistical information about MPI functions. mpiP is a link-time library, that means that it can be linked to the object file, though it is recommended that you have recompiled the code with -g. Debugging information is used to decode the program counters to a source code filename and line number automatically. mpiP will work without -g, but mileage may vary.

Usage example:

```bash
# assume you have you MPI flavor module loaded
module load mpip/3.4.1

# link or compile your code from scratch with -g
mpif90 -g -o my_app my_app.f90 -lm -lbfd -liberty -lunwind
# or
mpif90 -o my_app my_app.o -lm -lbfd -liberty -lunwind

# run the code normally (either interactively with salloc or as usual with sbatch)
salloc -p play --ntasks=4 srun mpi_app
```

If everything works, you will see the mpiP header preceding your program stdout, and there will be generated a text report file in your work directory. File is small, no worries about quota. Please, consult the link above for the
file content explanation. During runtime, one can set MPIP environment variables to change the profiler behavior. Example:

```bash
export MPIP="-t 10.0 -k 2"
```

**Scalasca**

Available through module load scalasca

**Quotas**

Triton has quotas which limit both the space usage and number of files. The quota for your home directory is 10GB, for $WRKDIR by default is 200GB, and project directories depending on request. These quotas exist to avoid usage exploding without anyone noticing. If you ever need more space, just ask. We’ll either give you more or find a solution for you.

The file quota is because scratch is not that great for too many small files. If you have too many small files, see the page on small files.

Normally, things just work, but there are certain intrinsic problems in scratch, so if you ever get a “disk quota exceeded” error, then read on.

**Note:** To try a quick fix for scratch, you can: `quotafix -gs --fix /path/to/the/directory`

If that fixes something, and problem recurs, then: `module load teflon`

To check your home directory usage, run: `du -xh $HOME | sort -h`

**How quotas work**

There are both quotas for users and projects (/m/$dept/$project). However, Lustre (scratch) can currently only do quotas by user or group, not by file path. If you `ls -l` a file or directory, you see both user and group. Unfortunately, with lustre it only really works for one of them at a time. So, on Triton, we use user private groups: everyone has a group with the same name as your user, and in $WRKDIR all files should have your group, and in project directories the group of that project. We have things set up so that things will Just Work if you do normal things.

```bash
$ ls -l test
drwxrwsr-x 3 darstr1 darstr1 4096 Jan 25 15:13 test/
  ^^^^^^^  ^^^^^^^
  |-- user  |-- group
  ^-- SETGID
```

**Important!** If a file has a group of domain users or triton-users, which occurs by default, then there is no quota for the files! To get around this, we have all directories “SETGID” (chmod g+s) and then files automatically are made in the correct group. That leads to the next point…
Disk quota exceeded error but I have plenty of space!

If the `quota` command says you have plenty of space AND sufficient number of files, then you’ve hit a common problem. Probably, the directory does not have the SETGID bit set, so when you try to make a new file, it appears as group ‘domain users’, and there’s no quota assigned, so it fails!

**Quick fixes**

Run either quotafix which will try to do things automatically, or this find command. You can only fix $WRKDIR on Triton, since the user-private-group does not exist on Aalto Linux workstations.

```bash
# AUTOMATIC ON TRITON: Fix everything.
# (only for $WRKDIR or group directories, still in testing):
/share/apps/bin/quotafix -sg --fix /path/to/dir/

# MANUAL ON TRITON: use find yourself.
# $GROUP is your username for work, or project-group name for scratch.
lfs find /path/to/dir -type d -print0 | xargs -0 chmod g+s
lfs find /path/to/dir ! -group $GROUP -print0 | xargs -0 chgrp $GROUP

# AALTO WORKSTATIONS: use "find" instead of "lfs find" above.
```

**Details**

Check the SETGID bit and group ownership for directories: `drwxrwsr-x`. Directories must have “s” there and the right group, otherwise when you try to make new files in that directory, they are group= ‘domain users’ and it fails.

**I can’t rsync/sftp/etc**

It is related to the above mentioned issue, something like rsync -a ... or cp -p ... are trying to save original group ownership attribute, which will not work. Try this instead:

```bash
## mainly one should avoid -g (as well as -a) since it preserves the old group (with no quota)
$ rsync -urlptDxv --chmod=Dg+s somefile triton.aalto.fi:/path/to/work/directory

## avoid '-p' with cp, or if you want to keep timestapms, mode etc, then use '--preserve=
$ cp -r --preserve=mode,timestamps somefile /path/to/mounted/triton/work/directory
```

You may need similar things for other different programs.

*Details:* Some programs change the group or don’t preserve the SETGID bit. This especially happens when you try to copy a directory from somewhere else to Triton while preserving the SETGID bit. You get a directory in the wrong group, or directory without SETGID bit so new files are in the wrong group, so no quota.
Other solutions

teflon

This is a new hack we are working on and hasn’t been extensively tested. Teflon is “anti-SETGID” which stops any program from changing either the group or SETGID bit, using LD_PRELOAD magic. It should work with any program, currently probably only 64-bit though. This is still under development. Please report problems or success stories.

You have to run quotafix or chmod/chgrp commands above first.

```bash
# Use via a module - applies to everything in this session.
module load teflon

# OR: Run a single program under teflon
/share/apps/bin/teflon your_program [args]] ]>newgrp
→You can do the below commands, and it will change your default group. This is per-shell (it makes a subshell). When you are done, use exit to revert back.
→Theoretically there's no downside to this, but if you alternate between project_
→directories and group directories, eventually the quotas will get mixed up between_
→the directories.newgrp $USER

newgrp $PROJECT_GROUP  # for project directories - find the right group
```

Singularity Containers

For more information see: https://sylabs.io/docs

Basic Idea

The basic idea behind Singularity containers is that software is packaged into a container (basically an entire self-contained operating system in a file!) that can be based on a Docker image that can then be run by the user. This allows hard to install software to be easily packaged and used - because you are packaging the entire OS!

During runtime, the root file system / is changed to the one inside the image and file systems are brought into the container through bind mounts. Effectively, the programs in the container are run in an environment mostly defined by the container image, but the programs can read and write specific files in Triton. Typically, e.g. the home directory comes from Triton.

This sounds complicated, but in practice this is easy due to singularity_wrapper written for Triton. You can also run singularity on triton without the wrapper, but you may need to e.g. bind /scratch yourself to access your data.
Basic Usage with singularity_wrapper

While the image itself is read-only, remember that /home, /m, /scratch and /l etc. are not. If you edit/remove files in these locations within the image, that will happen outside the image as well.

On Triton, you just need to load the proper module. This will set some environment variables and enable the use of singularity_wrapper.

singularity_wrapper is written so that when you load a module written for a singularity image, all the important options are already handled for you. It has three basic commands:

1. `singularity_wrapper shell <shell>` - Gives user a shell within the image (specify `<shell>` to say which shell you want).
2. `singularity_wrapper exec <cmd>` - Executes a program within the image.
3. `singularity_wrapper run <parameters>` - Runs the singularity image. What this means depends on the image in question - each image will define a “run command” which does something. If you don’t know what this is, use the first two instead.

Under the hood, singularity_wrapper does this:

1. Choosing appropriate image based on module version
2. Binding of basic paths (-B /l:/l, /m:/m, /scratch:/scratch)
3. Loading of system libraries within images (if needed) (e.g. -B /lib64/nvidia:/opt/nvidia)
4. Setting working directory within image (if needed)

Short guide to Singularity commands

These are the “raw” singularity commands. If you use these, you have to configure the images and bind mounts yourself (which is done automatically by singularity_wrapper). If you module show the module you can get hints about what happens.

Singularity enables three base commands to user:

1. `singularity shell <image>` - Gives user a shell within the image (see singularity shell --help for more information on flags etc.)
2. `singularity exec <image> <cmd>` - Executes a program within the image (see singularity exec --help for more information on flags etc.)
3. `singularity run <image> <parameters>` - Runs the singularity image. What this means depends on the image in question. (see singularity run --help for more information on flags etc.)

An example use case with MPI in singularity

The Serpent code is a Hybrid MPI/OpenMP particle following code, and can be installed into a container using the definition file sss2.def, which creates a container based on Ubuntu v. 20.04. In the build process, Singularity clones the Serpent source code, installs the required compilers and libraries, including the MPI library to the container. Furthermore, datafiles needed by Serpent are included in the container. Finally, a python environment with useful tools are also installed into the container. The Serpent code is compiled and the executable binaries are saved and the source code is removed.

The container can be directly used with the Triton queue system assuming the datafiles are stored in the user home folder. The file sss2.slurm.cmd can be used as an example. If scratch is used, please add -B /scratch after “exec” in the file.
The key observations to make:

1. `mpirun` is called in Triton, which launches multiple Singularity containers (one for each MPI task). Each container directly launches the `sss2`-executable. Each container can run multiple OpenMP threads of Serpent.

2. The openMPI library (v. 4.0.3) shipping with Ubuntu 20.04 seems to be compatible with the Triton module `openmpi/4.0.5`.

3. The Ubuntu MPI library binds all the threads to the same CPU. This is avoided by passing the parameter `--bind-to none` to `mpirun`.

4. The infiniband is made available by the `mpirun` parameter `--mca btl_openib_allow_ib`.

Small files

Millions of small files are a huge problem on any filesystem. You may think `/scratch`, being a fast filesystem, doesn’t have this problem, but it’s actually worse here. Lustre (scratch) as like an object store, and stores files separately from metadata. This means that each file access requires multiple different network requests, and making a lot of files brings your research (and managing the cluster) to a halt. What counts as a lot? Your default quota is 1e6 files. 1e4 for a project is not a lot. 1e6 for a single project is.

You may have been directed here because you have a lot of files. In that case, welcome to the world of big data, even if your total size isn’t that much! (it’s not just size, but difficulty of handling using normal tools) Please read this and see what you can learn, and ask us if you need help.

This page is mostly done, but specific examples could be expanded.

See also:

- Data storage on the Lustre file system, especially the bottom.
- Compute node local drives

Contents

The problem with small files

You know Lustre is high performance and fast. But, there is a relatively high overhead for accessing each file. Below, you can see some sample transfer rates, and you can see that total performance drops drastically when files get small. (These numbers were for the pre-2016 Lustre system, it’s better now but the same principle applies.) This isn’t just a problem when you are trying to read files, it’s also a problem when managing, moving, migrating, etc.

<table>
<thead>
<tr>
<th>File size</th>
<th>Net transfer rate, many files of this size</th>
</tr>
</thead>
<tbody>
<tr>
<td>10GB</td>
<td>1100 MB/s</td>
</tr>
<tr>
<td>100MB</td>
<td>990 MB/s</td>
</tr>
<tr>
<td>1MB</td>
<td>90MB/s</td>
</tr>
<tr>
<td>10KB</td>
<td>.9MB/s</td>
</tr>
<tr>
<td>512B</td>
<td>.04 MB/s</td>
</tr>
</tbody>
</table>
Why do people make millions of small files?

We understand there reasons people make lots of files: it’s convenient. Here are some of the common problems (and alternative solutions) people may be trying to solve with lots of files.

- Flat files are universal format. If you have everything in its own file, then any other program can look at any data individually. It’s convenient. This is a fast way to get started and use things.
- Compatibility with other programs. Same as above.
- Ability to use standard unix shell tools. Maybe your whole preprocessing pipeline is putting each piece of data in its own file and running different standard programs on it. It’s the Unix way, after all. Using filesystem as your index. Let’s say you have a program that reads/writes data which is selected by different keys. It needs to locate the data for each key separately. It’s convenient to put all of these in their own files: this takes the role of a database index, and you simply open the file with the name of the key you need. But the filesystem is not a good index.
  - Once you get too many files, a database is the right tool for the job. There are databases which operate as single files, so it’s actually very easy.
- Concurrency: you use filesystem as the concurrency layer. You submit a bunch of jobs, each job writes data to its own file. Thus, you don’t have to worry about problems with appending to the same file/database synchronization/locking/etc. This is actually a very common reason.
  - This is a big one. The filesystem is the most reliable way to join the output of different jobs (for example an array job), and it’s hard to find a better strategy. It’s reasonable to keep doing this, and combine job outputs in a second stage to reduce the number of files
- Safety/security: the filesystem isolates different files from each other, so if you modify one, there’s less chance of corrupting any other ones. This goes right along with the reason above.
- You only access a few files at a time in your day to day work, so you never realize there’s a problem. However, when we try to manage data (migrate, move, etc), then a problem comes up.
- Realize that forking processes has similar overhead. Small reads are also non-ideal, but less bad(?).

Strategies

- Realize you will have to have to change your workflow. You can’t do everything with grep, sort, wc, etc. anymore. Congratulations, you have big data.
- Consider right strategy for your program: a serious program should provide options for this.
  - For example, I’ve seen some machine learning frameworks which provide an option to compress all the input data into a single file that is optimized for reading. This is precisely designed for this type of case. You could read all the files individually, but it’ll be slower. So in this case, one should first read the documentation and see there’s a solution. One would take all the original files and make the processed input files. Then, take the original training data, package it together in one compressed archive for long-term storage. If you need to look at individual input files, you can always decompress one by one.
- Split - combine - analyze
  - Continue like you have been doing: each (array?) job makes different output files. Then, after running, combine the outputs into one file/database. Clean up/archive the intermediate files. Use this combined DB/file to analyze the data in the long term. This is perhaps the easiest way to adapt your workflow.
- HDF5: especially for numerical data, this is a good format for combining your results. It is like a filesystem within a file, you can still name your data based on different keys for individual access.
- Unpack to local disk, pack to scratch when done.
– Main article: *Compute node local drives*,

– This strategy can be combined with many of the other strategies below

– This strategy is especially good when your data is write-once-read-many. You package all of your original data into one convenient archive, and unpack it to the local disk when you need it. You delete it when you are done.

- Use a proper database suitable for your domain (sqlite): Storing lots of small data where anything can be quickly findable and you can do computation efficiently is exactly what databases do. It can be difficult to have a general purpose database work for you, but there are a wide variety of special-purposes databases these days. Could one of them be suitable for storing the results of your computation for analysis?

  – Note that if you are really doing high-performance random IO, putting a database on scratch is not a good idea, and you need to think more.

  – Consider combining this with local disk: You can copy your pre-created database file to local disk and do all the random access you need. Delete when done. You can do modification/changes directly on scratch if you want.

- key-value stores: A string key stores arbitrary data.

  – This is a more general database, basically. It stores arbitrary data for a certain key.

- Read all data to memory.

  – A strategy for using many files. Combine all data into one file, read them all into memory, then do the random access in memory.

- Compress them down when done.

  – It’s pretty obvious: when you are done with files, compress all of them into one. You have the archive and can always unpack when needed. You should especially at least do this when you are done with a project: if everyone did this, the biggest problems could be solved.

- Make sure you have proper backups for large files, mutating files introduces risks!

  – If you do go using these strategies, make sure you don’t accidentally lose something you need. Have backups (even if it’s on scratch: backup your database files)

- If you do have to keep many small files, check the link above for lustre performance tuning.

  – *Data storage on the Lustre file system*

- If you have other programs that can only operate on separate files

  – This is a tough situation, investigate what you can do combining the strategies above. At least you can pack up when done, and possibly copying to local disk while you are accessing is a good idea.

- MPI-I/O: if you are writing your own MPI programs, this can parallelize output

**Specific example: HDF5 for numerical data, or some database**

HDF5 is essentially a database for numerical data. You open a HDF5 file and access different data by path - the path is like a filename. There are libraries for accessing this data from all relevant programming languages.

If you have some other data that is structured, there are other databases that will work. For example, sqlite is a single-file, serverless database for relational data, and there are other similar things for time serieses or graphs.
Specific example: Unpacking to local disk

You can see examples at compute node local drives

Specific example: Key-value stores

Let’s say you have written all your own code and want an alternative to files. Instead, use a key-value database. You open one file, and store your file contents under different keys. When you need the data out, you request it by that key again. The keys take the place of filenames. Anytime you would open files, you just access from these key-value stores. You also have ways of dumping and restoring the data if you need to analyze it from different programs.

Performance tuning for small files

See here: Data storage on the Lustre file system

Triton ssh key fingerprints

ssh key fingerprints allow you to verify the server you are connecting to. The usual security model is that once you connect once, you save the key and can always be sure you are connecting to the same server from then on. To be smarter, you can actually verify the keys the first time you connect - thus, they are provided below.

You can verify SSH key fingerprints with a command like:

```
ssh-keygen -l -E sha256 -f <(ssh-keyscan triton.aalto.fi 2>/dev/null)
```

Here are the SSH key fingerprints for Triton:

```
256 SHA256:04Wt813WFsYj27KiAyo3u6RiGBelq1R19oJd2GXIaho no comment (ECDSA)
256 SHA256:1Mj2Gpf6iInwnY/Yf9g/b/wToaUaOU87zzzCtibj6I no comment (ED25519)
2048 SHA256:glizQJUQKoGcN2aTtp9jTxuJjTtnrKxRD8yImE06RJQ no comment (RSA)
```

and the same but with md5 hashes:

```
256 MD5:ac:61:86:86:el:11:29:f5:46:23:d8:25:00:8a:7b:00 no comment (ECDSA)
2048 MD5:1d:73:89:ae:3c:5a:5a:2a:04:76:cc:06:6a:79:90 no comment (RSA)
```

Or this can be copied and pasted directly into your .ssh/authorized_keys file:

```
triton.aalto.fi ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAADBQk8MvTSB2gYf9Y969vhMczdGSO+rnGZQhZLUGMkMduq4q+b/
LpHcn/yH1JNY8WEIDt8NELdn1+/9umk/
zk7IhxtnPvN2uAYO1t1Hh7Kk7zQF0eHgmYcPHSSDf12XENY6cQ1gHeFaF4QT483+f9fvU1p7E+MKQIz3dKreKm4AHdTkChj
zDQq9qgodg3gCznM+6YoaGyf4wICJMjKUBQP0mug5FYWX43StaPh+hcOQFYOiKjOyEBY/
HXOuDsZgCCG2b9qWhTrA3svcSKK46X76sXOR+8Ftb7u9xnLgm+903+zsGfsEQY2enXrf7YChNz4y5ASF
```

There is also a page for ssh host keys for the Aalto shell servers kosh, lyta, brute, force

6.1. Triton cluster
Storage

See also:

The storage tutorial is a prerequisite.

These pages are also related and include solutions to common storage problems:

- Storage: Lustre (scratch)
- Storage: local drives
- Quotas
- Small files

This pages gives an overview of more advanced storage topics. You should read the storage tutorial first.

Checklist

Do any of these apply to you? If so, consider your situation and ask us for help!

If you have been sent this checklist because your jobs may be having a lot of IO, don’t worry. It’s not necessarily a problem but please go through this checklist and let us know what applies to you so we can give some recommendations.

- Many small files being accessed in jobs (hundred or more).
- Files with extremely random access, in particular databases or database-like things (hdf5).
- Files being read over and over again. Alternatives: copy to local disks, or read once and store in memory.
- Number of files growing, for example all your runs have separate input files, output files, Slurm output files, and you have many runs.
- Constantly logging to certain files, writing to files from many parallel jobs at the same time.
- Reading from single files from many parallel jobs or threads at the same time.
- Is all your IO concatenated at one point, or spread out over the whole job?

(and if we’ve asked you specifically about your jobs, could you also describe what kind of job it is, the type of disk read and write happens, and in what kinds of pattern? Many small files, a few large ones, reading same files over and over, etc. How’s it spread out across jobs?)

If you think your IO may have bad patterns or even you just want to talk to make sure, let one of the Triton staff know or submit an issue in the issue tracker.

Checking your stats

You can monitor your jobs’ Lustre usage with our jobstats-utility.

You can also check the total disk read and write of your past jobs using:

```
# All your recent jobs:
sacct -o jobid%10,user%8,jobname%10,NodeList,MaxDiskRead,MaxDiskWrite -u $USER
# A single jobid
sacct -o jobid%10,user%8,jobname%10,NodeList,MaxDiskRead,MaxDiskWrite -j $jobid
```

These statistics are calculated on the whole node and thus include IO caused by other jobs on the same server while your job is running. For more detailed stats, use jobstats.
Loading data for machine learning

As we’ve said before, modern GPUs are super data-hungry when used for machine learning. If you try to open many files to feed it the data, “you’re going to have a bad time”. Luckily, different packages have different solutions to the problem.

In general, at least try to combine all of your input data into some sort of single file that can be read in sequence.

Try to do the least amount of work possible in the core training loops: any CPU usage, print, logging, preprocessing, postprocessing, etc. reduces the amount of time the GPU is working unless you do it properly (Amdhal’s law).

- Tensorflow: data input pipelines
(more coming later)

Compilers and toolchains

Individual compilers

Some of our compilers are available as individual modules. If you simply want a compiler, we recommend you use these modules.

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>5.5.0</td>
</tr>
<tr>
<td>gcc</td>
<td>6.3.0</td>
</tr>
<tr>
<td>llvm</td>
<td>6.0.1-python2</td>
</tr>
<tr>
<td>llvm</td>
<td>6.0.1-python3</td>
</tr>
</tbody>
</table>

If you need libraries or tools (e.g. openmpi or cmake) to compile your software, use modules with lower-case names. These modules are individual modules that do not have extra toolchain requirements.

Toolchains

Some modules in Triton are organized in so-called toolchains. These are collections of compilers and tools that are used for compiling specialized software.

Typically a toolchain contains a compiler and a MPI implementation, but it can also contain additional mathematical and computational libraries.

Naming convention is from EasyBuild that is used to administer the software collections. It goes like: <compiler><mpi><blas><lapack><fftw><cuda>

eg. GCC,OpenMPI,OpenBLAS,LAPACK,FFTW,CUDA would result in toolchain goolfc
Toolchains in detail

<table>
<thead>
<tr>
<th>Toolchain</th>
<th>Compiler version</th>
<th>MPI version</th>
<th>BLAS version</th>
<th>ScaLA-PACK version</th>
<th>FFTW version</th>
<th>CUDA version</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goolf Toolchains:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>goolf/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>Open-MPI/1.10.2</td>
<td>Open-BLAS/0.2.15</td>
<td>ScaLA-PACK/2.0.2</td>
<td>FFTW/3.3.4</td>
<td></td>
</tr>
<tr>
<td>goolf/triton-2016b</td>
<td>GCC/5.4.0</td>
<td>Open-MPI/1.10.3</td>
<td>Open-BLAS/0.2.18</td>
<td>ScaLA-PACK/2.0.2</td>
<td>FFTW/3.3.4</td>
<td>*</td>
</tr>
<tr>
<td>goolfc/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>Open-MPI/1.10.2</td>
<td>Open-BLAS/0.2.15</td>
<td>ScaLA-PACK/2.0.2</td>
<td>FFTW/3.3.4</td>
<td>7.5.18</td>
</tr>
<tr>
<td>goolfc/triton-2017a</td>
<td>GCC/5.4.0</td>
<td>Open-MPI/2.0.1</td>
<td>Open-BLAS/0.2.19</td>
<td>ScaLA-PACK/2.0.2</td>
<td>FFTW/3.3.4</td>
<td>8.0.61</td>
</tr>
<tr>
<td><strong>Gmpolf Toolchains:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmpolf/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>MPICH/3.0.4</td>
<td>Open-BLAS/0.2.15</td>
<td>ScaLA-PACK/2.0.2</td>
<td>FFTW/3.3.4</td>
<td>*</td>
</tr>
<tr>
<td>gmpolfc/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>MPICH/3.0.4</td>
<td>Open-BLAS/0.2.15</td>
<td>ScaLA-PACK/2.0.2</td>
<td>FFTW/3.3.4</td>
<td>7.5.18</td>
</tr>
<tr>
<td><strong>Gmvolf Toolchains:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gmvolf/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>MVA-PICH2/2.0.1</td>
<td>Open-BLAS/0.2.15</td>
<td>ScaLA-PACK/2.0.2</td>
<td>FFTW/3.3.4</td>
<td>*</td>
</tr>
<tr>
<td>gmvolfc/triton-2016a</td>
<td>GCC/4.9.3</td>
<td>MVA-PICH2/2.0.1</td>
<td>Open-BLAS/0.2.15</td>
<td>ScaLA-PACK/2.0.2</td>
<td>FFTW/3.3.4</td>
<td>7.5.18</td>
</tr>
<tr>
<td><strong>Ioolf Toolchains:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ioolf/triton-2016a</td>
<td>icc/2015.3.187</td>
<td>Open-MPI/1.10.2</td>
<td>Open-BLAS/0.2.15</td>
<td>ScaLA-PACK/2.0.2</td>
<td>FFTW/3.3.4</td>
<td>*</td>
</tr>
<tr>
<td><strong>Iomkl Toolchains:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iomkl/triton-2016a</td>
<td>icc/2015.3.187</td>
<td>Open-MPI/1.10.2</td>
<td>imkl/11.3.1.150</td>
<td>imkl/11.3.1.150</td>
<td>imkl/11.3.1.150</td>
<td>*</td>
</tr>
<tr>
<td>iomkl/triton-2016b</td>
<td>icc/2015.3.187</td>
<td>Open-MPI/1.10.3</td>
<td>imkl/11.3.1.150</td>
<td>imkl/11.3.1.150</td>
<td>imkl/11.3.1.150</td>
<td>*</td>
</tr>
</tbody>
</table>

Some of our software is compiled against these toolchains and we update them to newer versions if needed. If you require older versions of e.g. GCC we will install them as individual modules.

When asking for specialized software, these will be used as the starting point. E.g. Armadillo/6.700.3-goolf-triton-2016a-Python-2.7.11 uses goolf-triton-2016a as the base.
Remote workflows at Aalto

Note: The more specific remote access instructions for scicomp is at Remote Access (recent email had duplicate links to this page). This page explains the options, including other systems.

Note: How can you work from home? For that matter, how can you work on more than your desktop/laptop while at work? There are many options which trade off between graphical interfaces and more power. Read more for details.

You have most likely created your own workflow to analyse data at Aalto and most likely you are using a dedicated desktop workstation in Otaniemi. However, with increased mobility of working conditions and recent global events that recommend tele-working, you might be asking yourself: “how do I stop using my workstation at the dept, and get analysis/figures/papers done from home?”

The data analysis workflows from remote might not be familiar to everyone. We list here few possible cases, this page will expand according to the needs and requests of the users.

What’s your style?

If you need the most power or flexibility, use Triton for your data storage and computation. To get started, you can use Jupyter (4) and VDI (3) which are good for developing and prototyping. Then to scale up, you can use the Triton options: 6, 7, 8 which have access to the same data. (Triton account required for 4-8).

If you need simple applications with a graphical interface, try 3 (VDI).

If you use your own laptop/desktop (1, 2), then it’s good for getting started but you have to copy your data and code back and forth once you need to scale up.

Summary table for remote data analysis workflows

- Good for data security: 3, 4, 5, 6, 7
- Good for prototyping, working on the go, doing tests, interactive work: 1, 2, 3, 4, 5
- Shares Triton data (e.g. scratch folders): 3, 4, 5, 6, 7
- Easy to scale up, shares software, data, etc: 4, 5, 6, 7
- Largest resources available 7 (medium: 6)
Fig. 1: <Overview of all the computing options at Aalto University>
<table>
<thead>
<tr>
<th>Workflow</th>
<th>Pros</th>
<th>Cons</th>
<th>Recommendation</th>
<th>Triton data Y/N</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1. Own laptop/desktop computer</strong></td>
<td>Can work from anywhere. Does not require internet connection. You are in control.</td>
<td>Not good for personal or confidential data. Computing resources might not be enough. Accessing large data remotely stored at Aalto might be problematic - you will end up having to copy a lot. You have to manage software yourself.</td>
<td>Excellent for prototyping, working on the go, doing tests, interactive work (e.g. making figures). Don’t use it with large data or confidential / personal data.</td>
<td>N</td>
</tr>
<tr>
<td><strong>2. Aalto laptop</strong></td>
<td>Same as above, plus same tools available as Aalto employer.</td>
<td>Same as above.</td>
<td>Same as above.</td>
<td>N</td>
</tr>
<tr>
<td><strong>3. Remote virtual machine (<a href="https://vdi.aalto.fi">https://vdi.aalto.fi</a>)</strong></td>
<td>Computing happens on remote. Data access happens on remote, so it is more secure.</td>
<td>Computing resources are limited.</td>
<td>Excellent for prototyping, working on the go, doing tests, interactive work (e.g. making figures). More secure access to data.</td>
<td>Y</td>
</tr>
<tr>
<td><strong>4. Aalto Jupyter-hub (<a href="https://jupyter.triton.aalto.fi">https://jupyter.triton.aalto.fi</a>)</strong></td>
<td>Cloud based - resume work from anywhere. Includes command line (#6) and batch (#7) easily. Same data as seen on Triton (/scratch/dept/ and /work/ folders)</td>
<td>Jupyter can become a mess if you aren’t careful. You need to plan to scale up with #7 eventually, once your needs increase.</td>
<td>Excellent for prototyping, working on the go, doing tests, interactive work (e.g. making figures). Secure access to data. Use if you know you need to switch to batch jobs eventually (#7).</td>
<td>Y</td>
</tr>
<tr>
<td><strong>5. Interactive graphical session on Triton HPC (ssh -X)</strong></td>
<td>Graphical programs.</td>
<td>Lost once your internet connection dies, needs fast internet connection.</td>
<td>If you need specific graphical applications which are only on Triton.</td>
<td>Y</td>
</tr>
<tr>
<td><strong>6. Interactive command line session on Triton HPC (ssh + sinteractive)</strong></td>
<td>Works from anywhere. Can get lots of resources for a short time.</td>
<td>Limited time limits, must be used manually.</td>
<td>A general workhorse once you get comfortable with shell - many people work here + #7.</td>
<td>Y</td>
</tr>
<tr>
<td><strong>7. Non-interactive batch HPC computing on Triton (ssh + sbatch)</strong></td>
<td>Largest resources, bulk computing</td>
<td>Need to script your computation</td>
<td>When you have the largest computational needs.</td>
<td>Y</td>
</tr>
<tr>
<td><strong>6.1. Triton cluster</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>8. Non-interactive batch HPC computing on CSC</strong></td>
<td>Similar to #7 but at CSC</td>
<td>Similar to #7</td>
<td>Similar to #7</td>
<td>N</td>
</tr>
</tbody>
</table>
1. Own laptop/desktop computer

**Description:** Here you are the administrator. You might be working from a cafe with your own laptop, or from home with a desktop. You should be able to install any tool you need. As an Aalto employer you get access to many nice commercial tools for your private computers. Visit: https://download.aalto.fi/index-en.html and https://aalto.onthehub.com/ for some options.

**Pros:** Computing freedom! You can work anywhere, you can work when there is no internet connection, you do not share the computing resources with other users so you can fully use the power of your computer.

**Cons:** If you work with personal or confidential data, the chances of a data breach increase significantly, especially if you work from public spaces. Even if you encrypt your hard disks (links:https://www.aalto.fi/en/cyber-security-hub-under-construction/aalto-it-securitys-top-10-tips-for-daily-activities ) and even if you are careful, you might be forgetting to lock your computer or somebody behind you might see which password you type. Furthermore, personal computers have limited resources when it comes to RAM/CPUs/GPUs. When you need to scale up your analysis, you want to move it to an HPC cluster, rather than leaving scripts running for days. Finally, although you can connect your Aalto folders to your laptop (link https://scicomp.aalto.fi/aalto/remoteaccess.html and https://scicomp.aalto.fi/triton/tut/storage.html#accessing-and-transferring-files-remotely), when the data size is too big, it is very inefficient to analyse large datasets over the internet.

**Recommendation:** Own computer is excellent for prototyping data analysis scripts, working on the go, doing tests or new developments. You shouldn’t use this option if you are working with personal data or with other confidential data. You shouldn’t use this option if your computational needs are much bigger.

2. Aalto laptop

**Description:** As an Aalto employer, you are usually provided with a desktop workstation or with an Aalto laptop. With an Aalto laptop you can apply for administrator rights (link to the form) and basically everything you have read for option 1 above is valid also in this case. See “Aalto {Linux|Mac|Windows}” on scicomp’s Aalto section at https://scicomp.aalto.fi/aalto/.

**Pros/Cons/Recommendation:** see option 1 above. But, when on Aalto networks, you have easier access to Aalto data storage systems.

3. Remote virtual machine with VDI

**Description:** You might be working with very large datasets or with confidential/personal data, so that you cannot or do not want to copy the data to your local computer. Sometimes you use many computers, but would like to connect to “the same computer” from remote where a longer analysis script might be crunching numbers. Aalto has a solution called VDI https://vdi.aalto.fi (description at aalto.fi) where you can get access to a dedicated virtual machine from remote within the web browser. Once logged in, you can pick if you prefer Aalto Linux or Aalto Windows, and then you see the same interface that you would see if you logged in from an Aalto dedicated workstation. To access Triton data from the Linux one, use the path /m/{dept}/scratch/ (just like Aalto desktops).

**Pros:** The computing processes are not going to run on your local computer, computing happens on remote which means that you can close your internet connection, have a break, and resume the work where you left it. There is no need to copy the data locally as all data stays on remote and is accessed as if it was a desktop computer from the campus.

**Cons:** VDI machines have a limited computing power (2 CPUs, 8GB of RAM). So they are great for small prototyping, but for a large scale computation you might want to consider Aalto Triton HPC cluster. The VDI session is not kept alive forever. If you close the connection you can still resume the same session within 24h, after that you are automatically logged out to free resources for others. If you have a script that needs more than 24h, you might want to consider Aalto Triton HPC.
**Recommendation:** VDI is excellent when you need a graphic interactive session and access to large data or to personal/confidential data without the risks of data breach. Use VDI for small analysis or interactive development, we do not recommend it when the executing time of your scripts starts to be bigger than a 7 hours working day.

### 4. Aalto Jupyterhub

**Description:** Jupyter notebooks are a way of interactive, web-based computing: instead of either scripts or interactive shells, the notebooks allow you to see a whole script + output and experiment interactively and visually. They are good for developing and testing things, but once things work and you need to scale up, it is best to put your code into proper programs. Triton’s JupyterHub is available at https://jupyter.triton.aalto.fi. Read more about it at: https://scicomp.aalto.fi/triton/apps/jupyter.html. Triton account required.

**Pros:** JupyterHub it has similar advantages than #4, although data and code are accessed through the JupyterHub interface. In addition, things can stay running in the cloud. Although it can be used with R or Matlab, Python users will most likely find this to be a very familiar and comfortable prototyping environment. Similar to the VDI case, you can resume workflow (there are sessions of different lengths). You also also access Triton shell and batch (#6, #7) in the Jupyter interface, and it’s easy to scale up and use them all together.

**Cons:** You are limited to the Jupyter interface (but you can upload/download data, and integrate with many other things). Jupyter can become a mess if you aren’t careful. Computationally, an instance will always have limited CPUs and memory. Once you need more CPU/RAM, look into options #6 and #7 - they work seamlessly with the same data, software, etc.

**Recommendation:** Good for exploration and prototyping, access to large dataset, access to confidential/personal data. For more computational needs, be ready to switch to batch jobs (#7) once you are done prototyping.

### 5. Interactive graphical session on Triton HPC

**Description:** Sometimes what you can achieve with your own laptop or with VDI is not enough when it comes to computing resources. However, your workflow does not yet allow you to go fully automatic as you still need to manually interact with the analysis process (e.g. point-click analysis interfaces, doing development work, making figures, etc). An option is to connect to triton.aalto.fi with a graphical interface. This is usually done with ssh -X triton.aalto.fi. For example you can do it from a terminal within a VDI Linux session. Once connected to the triton log-in node, you can then request a dedicated interactive node with command sinteractive, and you can also specify the amount of CPU or RAM you need (link to sinteractive help page). Triton account required.

**Pros:** This is similar to the VDI case above (#3) without the computing limitation imposed by VDI.

**Cons:** If you connect from triton.aalto.fi from your own desktop/laptop, your internet connection might be limiting the speed of the graphical session making it very difficult to use graphical IDEs or other tools. Move to VDI, which optimises how the images are transferred over the internet. Sinteractive sessions cannot last for more than 24 hours, if you need to run scripts that have high computational requirements AND long time of execution, the solution for you is to go fully non-interactive using Triton HPC with slurm (case #6)

**Recommendation:** This might be one of the best scenarios for working from remote with an interactive graphical session. Although you cannot keep the session open for more than 24 hours, you can still work on your scripts/code/figures interactively without any limitation and without any risks of data breaches.
6. Interactive command line only session on Triton HPC/dept workstation

**Description**: sometimes you do not really need a graphical interface because you are running interactively scripts that do not produce or need a graphical output. This is the same case as sinteractive above, but without the limitation of the 24h session. The best workflow is to: 1) connect to triton `ssh triton.aalto.fi` 2) start a screen/tmux session that can be detached / reattached in case you lose the internet connection or in case you need to leave the interactive script running for days 3) request a dedicated interactive terminal with command `srun -p interactive --time=HH:MM:SS --mem=nnG --pty bash` (see other examples at https://scicomp.aalto.fi/triton/tut/interactive.html or https://scicomp.aalto.fi/triton/usage/gpu.html for interactive GPU) 4) get all your numbers crunched and remember to close it once you are done. Please note that, if you have a dedicated Linux workstation at a department at Aalto, you can also connect to your workstation and use it as a remote computing node fully dedicated to you. The resources are limited to your workstation, but here you won’t have the time constraint or the need to queue for resources if Triton’s queue is overcrowded. **Triton account required.**

**Pros**: when you do not need a graphical interface and when you need to run something interactively for days, this is the best option: high computing resources, secure access to data, persistent interactive session.

**Cons**: when you request an interactive command line session you are basically submitting a slurm job. As with all jobs, you might need to wait in the queue according to the amount of resources you have requested. Furthermore, jobs cannot last more than 5 days. In general, if you have an analysis script that needs more than 5 days to operate, you might want to identify if it can be parallelized or split into sub-parts with checkpoints.

**Recommendation**: this is the best option when you need long-lasting computing power and large data/confidential data access with interactive input from the user. This is useful once you have your analysis pipeline/code fully developed so that you can just run the scripts in command line mode. Post processing/figure making can then happen interactively once your analysis is over.

7. Non-interactive batch computing on Triton HPC

**Description**: this is the case when no interactive input is needed to process your data. This is extremely useful when you are going to perform the same analysis code for hundreds of time. Please check more detailed descriptions at https://scicomp.aalto.fi/triton/index.html and if you haven't, go through the tutorials https://scicomp.aalto.fi/triton/index.html#tutorials. **Triton account required.**

**Pros**: when it comes to large scale data analysis, this is the most efficient way to do it. Having a fully non-interactive workflow also makes your analysis reproducible as it does not require any human input which can sometimes be the source of errors or other irreproducible/undocumented steps.

**Cons**: as this is a non-interactive workflow, this is not recommended for generating figures or with graphical tools that does not allow “batch” mode operations.

**Recommendation**: this is the best option when you need long-lasting parallel computing power and large data/confidential data access. This is also recommended from reproducibility/replicability perspective since, by fully removing human input, the workflow can be made fully replicable.
8. Non-interactive batch HPC computing at CSC

**Description:** this case is similar to #7. You can read/learn more about this option at [https://research.csc.fi/guides](https://research.csc.fi/guides)

**Pro/Cons/Recommendation:** see #7.

### 6.1.5 Applications

See our *general information* and the full list below:

**Applications: General info**

See also:

Intro tutorial: *Applications* (this is assumed knowledge for all software instructions)

When you need software, check the following for instructions (roughly in this order):

- This page.
- Search the SciComp site using the search function.
- Check module spider and module avail to see if something is available but undocumented.
- The issue tracker for other people who have asked - some instructions only live there.

If you have difficulty, it’s usually a good idea to search the issue tracker anyway, in order to learn from the experience of others.

**Compilers**

See *Compilers and toolchains*.

**Modules**

See *Software modules*. Modules are the standard way of loading software.

**Singularity**

See *Singularity Containers*. Singularity are software containers that provide an operating system within an operating system. Software will tell you if you need to use it via Singularity.

**Software installation and policy**

We want to support all software, but unfortunately time is limited. In the chart below, we have these categories (which don’t really mean anything, but in the future should help us be more transparent about what we are able to support):

- **A:** Full support and documentation, should always work
- **B:** We install and provide best-effort documentation, but may be out of date.
- **C:** Basic info, no guarantees
If you know some application which is missing from this list but is widely in use (anyone else than you is using it) it would make sense install to /share/apps/ directory and create a module file. Send your request to the tracker. We want to support as much software as possible, but unfortunately we don’t have the resources to do everything centrally.

Software is generally easy to install if it is in Spack (check that package list page), a scientific software management and building system. If it has easy-to-install Ubuntu packages, it will be easy to do via singularity.

Software documentation pages

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
</tr>
<tr>
<td>A</td>
</tr>
</tbody>
</table>

**FHI-aims**

FHI-aims (Fritz Haber Institute ab initio molecular simulations package) is an electronic structure theory code package for computational molecular and materials science. FHI-aims density functional theory and many-body perturbation calculations at all-electron, full-potential level.

FHI-aims is licensed software with voluntary payment for an academic license. While the license grants access to the FHI-aims source code each holder of a license can use pre-built binaries available on Triton. To this end, contact Ville Havu at the PHYS department after obtaining the license.

On Triton the most recent version of FHI-aims is available via the modules FHI-aims/latest-intel-2020.0 that is compiled using the Intel Parallel Studio and FHI-aims/latest-OpenMPI-intel-2020.0-scalapack that is compiled without any Intel parallel libraries since in rare cases they can result in spurious segfaults. The binaries are available in /share/apps/easybuild/software/FHI-aims/<module name>/bin as aims.YYMMDD.scalapack.mpi.x where YYMMDD indicates the version stamp.

**Running FHI-aims on Triton**

To run FHI-aims on Triton a following example batch script can be used:

```
#!/bin/bash -l
#SBATCH --time=01:00:00
#SBATCH --constraint=avx # FHI-aims build requires at least AVX instruction set
#SBATCH --mem-per-cpu=2000M
#SBATCH --nodes=1
#SBATCH --ntasks=24

export OMP_NUM_THREADS=1
module load FHI-aims/latest-intel-2020.0
srun aims.YYMMDD.scalapack.mpi.x
```
Armadillo

supportlevel C

Armadillo http://arma.sourceforge.net/ is C++ linear algebra library that is needed to support some other software stacks. To get best performance using MKL as backend is advised.

The challenge is that default installer does not find MKL from non-standard location.

1. module load mkl
2. Edit “./build_aux/cmake/Modules/ARMA_FindMKL.cmake” and add MKL path to “PATHS”
3. Edit “./build_aux/cmake/Modules/ARMA_FindMKL.cmake” and replace mkl_intel_thread with mkl_sequential (we do not want threaded libs on the cluster)
4. Edit “include/armadillo_bits/config.hpp” and enable ARMA_64BIT_WORD
5. cmake . && make
6. make install DESTDIR=/share/apps/armadillo/<version>

Boost

supportlevel C

cangestupdated 2014

Boost is a numerical library needed by some other packages. There is a rpm-package of this in the default SL/RHEL repositories. In case the repository version is too old, a custom compilation is required.

To setup see the manual and follow the few simple steps to bootstrap and compile/install.

https://www.boost.org/doc/libs/1_56_0/more/getting_started/unix-variants.html

COMSOL Multiphysics

Hint: Join the other COMSOL users in our Zulip Chat: Stream “#triton”, topic “Comsol user group”.

To check which versions of Comsol are available, run:

module spider comsol

Comsol in Triton is best run in Batch-mode, i.e. without the graphical userinterface. Prepare your models on your workstation and bring the ready-to-run models to triton. However, various settings must be edited in the graphical user interface. For this, using vdi.aalto.fi to connect to triton is advisable.

For detailed tutorials from COMSOL, see for example the Comsol Knowledge base articles Running COMSOL® in parallel on clusters and Running parametric sweeps, batch sweeps, and cluster sweeps from the command line.
Prerequisites of running COMSOL in Triton

There is a largish but limited pool of floating COMSOL licenses in Aalto University, so please be careful not launch large numbers of comsol processes that each consume a separate license.

- Comsol uses a lot of temp file storage, which by default goes to $HOME. Fix a bit like the following:

  ```
  $ rm -rf ~/.comsol/
  $ mkdir /scratch/work/$USER/comsol_recoveries/
  $ ln -sT /scratch/work/$USER/comsol_recoveries/ ~/.comsol
  ```

- You may need to enable access to the whole filesystem in File|Options –> Preferences –> Security: File system access: “All files”

- Enable the “Study -> Batch and Cluster” as well as “Study -> Solver and Job Configurations” nodes in the “Show More Options dialog box you can open by right-clicking the study in the Model Builder Tree.

The cluster settings can be saved in consol settings, not in the model file. The correct settings are entered in File|Options –> Preferences –> Multicore and Cluster Computing. It is enough to choose Scheduler type: “SLURM”

You can test by loading from the Application Libraries the “cluster_setup_validation” model. The model comes with a documentation -pdf file, which you can open in the Application Libraries dialogue after selecting the model.
COMSOL requires MPICH2 compatible MPI libraries:

```bash
$ module purge
$ module load comsol/5.6 intel-parallel-studio/cluster.2020.0-intelmpi
```

**An example run in a single node**

Use the parameters `-clustersimple` and `-launcher slurm`. Here is a sample batch-job:

```bash
#!/bin/bash

# Ask for e.g. 20 compute cores
#SBATCH --time=10:00:00
#SBATCH --mem-per-cpu=2G
#SBATCH --cpus-per-task=20

cd $WRKDIR/my_comsol_directory
module load Java
module load comsol/5.6
module load intel-parallel-studio/cluster.2020.0-intelmpi

# Details of your input and output files
INPUTFILE=input_model.mph
OUTPUTFILE=output_model.mph

comsol batch -clustersimple -launcher slurm -inputfile $INPUTFILE -outputfile $OUTPUTFILE -tmpdir $TMPDIR
```

**Cluster sweep**

If you have a parameter scan to perform, you can use the Cluster sweep node. The whole sweep only needs one license even if comsol launches multiple instances of itself.

First set up the cluster preferences, as described above.

Start by loading the correct modules in triton (COMSOL requires MPICH2 compatible MPI libraries). Then open the graphical user interface to comsol on the login node and open your model.

```bash
$ module purge
$ module load comsol/5.6 intel-parallel-studio/cluster.2020.0-intelmpi
$ comsol
```

Add a “Cluster Sweep” node to your study and a “Cluster Computing” node into your “Job Configurations” (You may need to first enable them in the “Show more options”. Check the various options. You can try solving a small test case from the graphical user interface. You should see COMSOL submitting jobs to the SLURM queue.

For a larger run, COMSOL can then submit the jobs with comsol but without the GUI:

```bash
$ comsol batch -inputfile your_ready_to_run_model.mph -outputfile output_file.mph -study std1 -mode desktop
```

See also how to run a parametric sweep from command line?

Since the sweep may take some time to finnish, please consider using tmux or screen to keep your session open.
MATLAB + COMSOL – livelink

It is possible to control COMSOL with MATLAB. The blog post by KnifeLee was useful in preparation of this example.

Save a username and password for COMSOL mph server

Before your first use, you need to save the username and password for COMSOL mph server. On the login node, run:

```bash
$ module load comsol/5.6
$ comsol mphserver
```

And COMSOL will ask for you to choose a username and password. You can close the comsol server with “close”. Please note, that each instance of the below process uses a COMSOL licence, so this method is not useful for parameter scans.

Example files for batch job workflow

Here is an example batch submit script `comsol_matlab_livelink.sh`:

```bash
#!/bin/bash
#SBATCH --time=10:00:00
#SBATCH --nodes=1
#SBATCH --exclusive
module load matlab
module load comsol/5.6

echo starting comsol server in the background
comsol mphserver &
echo comsol is now running
matlab -nodesktop -nosplash -r "runner;exit(0)"
echo matlab closed
```

The MATLAB process is running the `runner.m` script:

```matlab
disp('Including comsol routines into the path. ');
addpath /share/apps/comsol/5.6/mli/
disp('Connecting to COMSOL from MATLAB')
mphstart(2036)
disp('Connection established')
disp('Starting Model Control Script')
```

(continues on next page)
disp('Exiting Matlab')
exit(0);

The Model Control Script `script.m` could be e.g. the following:

```matlab
import com.comsol.model.*;
import com.comsol.model.util.*;
model = ModelUtil.create('Model1');
model.component.create('comp1', true);
\...
```

The job is submitted with:

```
$ sbatch comsol_matlab_livelink.sh
```

**Cluster computing controlled from your windows workstation**

The following example shows a working set of settings to use triton as a remote computation cluster for COMSOL.

**Prerequisites:**

- Store ssh-keys in pagent so that you can connect to triton with putty without entering the password.
- Save / install **putty executables** locally, e.g. in Z:\putty:
  - plink.exe
  - pscp.exe
  - putty.exe
In this configuration, *s jjamsa* is replaced with your username.
Deep learning software

This page has information on how to run deep learning frameworks on Triton GPUs.

Theano

Installation

The recommended way of installing theano is with an anaconda environment.

Detectron

Detectron uses Singularity containers, so you should refer to that page first for general information. Detectron-image is based on a Dockerfile from Detectron’s repository. In this image Detectron has been installed to /detectron.

Usage

This example shows how you can launch Detectron on a gpu node. To run example given in Detectron repository one can use the following Slurm script:

```bash
#!/bin/bash
#SBATCH --time=00:30:00
#SBATCH --mem=8G
#SBATCH --gres=gpu:teslap100:1
#SBATCH -o detectron.out
module load singularity-detectron
mkdir -p $WRKDIR/detectron/outputs
singularity_wrapper exec python2 /detectron/tools/infer_simple.py \
   --cfg /detectron/configs/12_2017_baselines/e2e_mask_rcnn_R-101-FPN_2x.yaml \
   --output-dir $WRKDIR/detectron/outputs \
   --image-ext jpg \
   --wts https://s3-us-west-2.amazonaws.com/detectron/35861858/12_2017_baselines/e2e_\n   →mask_rcnn_R-101-FPN_2x.yaml.02_32_51.SgT4y1cO/output/train/coco_2014_train:coco_\n   →2014_valminusminival/generalized_rcnn/model_final.pkl \
   /detectron/demo
```

Now example can by run on GPU node with:

```bash
sbatch detectron.slrm
```

In typical usage one does not want to download models for each run. To use stored models one needs to:

1. Copy detectron sample configurations from the image to your own configuration folder:

   ```bash
   module load singularity-detectron
   mkdir -p $WRKDIR/detectron
   singularity_wrapper exec cp -r /detectron/configs $WRKDIR/detectron/configs
   cd $WRKDIR/detectron
   ```
2. Create data directory and download example models there:

```bash
mkdir -p data/ImageNetPretrained/MSRA
wget -O data/ImageNetPretrained/MSRA/R-101.pkl \
```

4. Edit the weights-parameter in configuration file `12_2017_baselines/e2e_mask_rcnn_R-101-FPN_2x.yaml`:

```bash
33c33
---
```

5. Edit Slurm script to point to downloaded weights and models:

```bash
#!/bin/bash
#SBATCH --time=00:30:00
#SBATCH --mem=8G
#SBATCH --gres=gpu:teslap100:1
#SBATCH -o detectron.out

module load singularity-detectron
mkdir -p $WRKDIR/detectron/outputs

singularity_wrapper exec python2 /detectron/tools/infer_simple.py \ 
  --cfg $WRKDIR/detectron/configs/12_2017_baselines/e2e_mask_rcnn_R-101-FPN_2x.yaml \ 
  --output-dir $WRKDIR/detectron/outputs \ 
  --image-ext jpg \ 
  --wts $WRKDIR/detectron/data/coco_2014_train:coco_2014_valminusminival/ \ 
  generalized_rcnn/model_final.pkl \ 
  /detectron/demo
```

6. Submit job:

```bash
sbatch detectron.slrm
```

**Fenics**

This uses *Singularity containers*, so you should refer to that page first for general information.

Fenics-images are based on these images.
Usage

This example shows how you can run a fenics example. To run example one should first copy the examples from the
image to a suitable folder:

```
mkdir -p $WRKDIR/fenics
cd $WRKDIR/fenics
module load singularity-fenics
singularity_wrapper exec cp -r /usr/local/share/dolfin/demo demo
```

The examples try to use interactive windows to plot the results. This is not available in the batch queue so to fix this
one needs to specify an alternative matplotlib backend. This patch file fixes example demo_poisson.py. Download it
into $WRKDIR/fenics and run

```
patch -d demo -p1 < fenics_matplotlib.patch
```

to fix the example. After this one can run the example with the following Slurm script:

```
#!/bin/bash
#SBATCH --time=00:15:00
#SBATCH --mem=1G
#SBATCH -o fenics_out.out
module purge
module load singularity-fenics
cd demo/documented/poisson/python/
srun singularity_wrapper run demo_poisson.py
```

To submit the script one only needs to run:

```
sbatch fenics.slrm
```

Resulting image can be checked with e.g.:

```
eog demo/documented/poisson/python/poisson.png
```

FMRIPrep

```
module load singularity-fmriprep/latest  # stick to the latest!
```

fmriprep is installed as a singularity container. By default it will always run the current latest version. If you need a
version that is not currently installed on triton, please open an issue at https://version.aalto.fi/gitlab/AaltoScienceIT/
triton/issues

Here an example to run fmriprep for one subject, using an interactive session, without free-surfer reconall, using
ica-aroma. The raw data in BIDS format are in the path <path-to-bids>, then you can create a folder for the
derivatives that is different than the BIDS folder <path-to-your-derivatives-folder>. Also create a temporary folder under your scratch/work folders for storing temporary files
<path-to-your-scratch-temporary-folder> for example /scratch/work/USERNAME/tmp/. The content of this folder is removed after fmriprep has finished.
# Example running in an interactive session

```bash
ssh triton.aalto.fi
sinteractive --time=24:00:00 --mem=20G  # you might need more memory or time depending

module load singularity-fmriprep/latest

singularity_wrapper exec fmriprep <path-to-bids> <path-to-your-derivatives-folder> --w
\n--path-to-your-scratch-temporary-folder participant --participant-label 01
--use-aroma --fs-no-reconall --fs-license-file /scratch/shareddata/set1/freesurfer/
\nlicense.txt
```

This might give the exit error “OSError: handle is closed”, this is a python thing, see https://neurostars.org/t/fmriprep-error-oserror-handle-is-closed/4030. The general rule is that if the reports are generated, then everything is done with the preprocessing.

If you want to parallelize things you can write a script that cycles through each subject labels and queues SBATCH jobs for each subject (it can be an array job or a series of serial jobs). It is important you tune your memory and time requirements before processing many subjects at once.

**POST-processing**

Fmriprep does the minimal preprocessing. There is no smoothing, no temporal filtering and in general you need to regress out the estimated confounds. The most simple way is:

```bash
module load fsl

fsl_regfilt -i $inputniifile -d "$file_with_bold_confounds.tsv" -o $outputniifile -f
\n-1,2,3,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
```

These are also tools for post-processing such as:

- https://github.com/HBClab/NuisanceRegression
- https://xcpengine.readthedocs.io/
- https://github.com/arielletambini/denoiser

These are not installed on the singularity image, hence you need to experiment with these on your own.

**Freesurfer**

```bash
module load freesurfer

Follow the instruction to source the init script specific to your shell.
FSL

module load fsl

Follow the instruction to source the init script specific to your shell.

GPAW

There is GPAW version installed in GPAW/1.0.0-goolf-triton-2016a-Python-2.7.11. It has been compiled with GCC, OpenBLAS and OpenMPI and it uses Python/2.7.11-goolf-triton-2016a as its base Python. You can load it with:

```
$ module load GPAW/1.0.0-goolf-triton-2016a-Python-2.7.11
```

You can create a virtual environment against the Python environment with:

```
$ export VENV=/path/to/env
$ virtualenv --system-site-packages $VENV
$ cd $VENV
$ source bin/activate
# test installation
$ python -c 'import gpaw; print gpaw'
```

GPAW site: https://wiki.fysik.dtu.dk/gpaw/

Gurobi Optimizer

Gurobi Optimizer is a commercial optimizing library.

License

Aalto University has a site-wide floating license for gurobi.

You can create a gurobi.lic file in your home folder. The file should contain the following single line:

```
TOKENSERVER=lic-gurobi.aalto.fi
```

The license is an Educational Institution Site License

Free Academic License Requirements, Gurobi Academic Licenses: Can only be used by faculty, students, or staff of a recognized degree-granting academic institution. Can be used for: Research or educational purposes. Consulting projects with industry – provided that approval from Gurobi has been granted.

Julia

The Julia programming language is a high-level, high-performance dynamic programming language for technical computing, in the same space as e.g. MATLAB, Scientific Python, or R. For more details, see their web page.
Interactive usage

Julia is available in the module system. By default the latest stable release is loaded:

```
module load julia
julia
```

Batch usage

Running Julia scripts as batch jobs is also possible. An example batch script is provided below:

```
#!/bin/bash
#SBATCH --time=00:01:00
#SBATCH --mem=1G
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK:-1}
module load julia
srun julia juliascript.jl
```

Number of threads to use

By default Julia uses up to 16 threads for linear algebra (BLAS) computations. In most cases, this number will be larger than the amount of CPUs reserved for the job. Thus when running Julia jobs it is a good idea to set the number of parallelization threads to be equal to the number of threads reserved for the job with `--cpus-per-task`. Otherwise, the performance of your program might be poor. This can be done by adding the following line to your slurm-script:

```
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK:-1}
```

Alternatively, you can use the `blas_set_num_threads()`-function in Julia.

Jupyter

**Note:** Quick link

Triton’s JupyterHub is available at https://jupyter.triton.aalto.fi.

**Note:** For new users

Are you new to Triton and want to access JupyterHub? Triton is a high-performance computing cluster, and JupyterHub is just one of our services - one of the easiest ways to get started. You still need a Triton account. This site has many instructions, but you should read at least:

- *About us, how to get help, and acknowledging Triton usage* (this JupyterHub is part of Triton, and thus Science-IT must be acknowledged in publications).
- *The accounts page*, in order to request a Triton account.
- Possibly the *storage page* to learn about the places to store data and how to transfer data.
- The JupyterHub section of this page (below).
If you want to use Triton more, you should finish the entire tutorials section.

Fig. 2: < Triton JupyterHub Demo >

Jupyter notebooks are a way of interactive, web-based computing: instead of either scripts or interactive shells, the notebooks allow you to see a whole script + output and experiment interactively and visually. They are good for developing and testing things, but once things work and you need to scale up, it is best to put your code into proper programs (more info). You must do this if you are going to large parallel computing.

Triton’s JupyterHub is available at https://jupyter.triton.aalto.fi. You can try them online at try.jupyter.org (there is a temporary notebook with no saving).

You can always run notebooks yourself on your own (or remote) computers, but on Triton we have some facilities already set up to make it easier.

How Jupyter notebooks work

- Start a notebook
- Enter some code into a cell.
- Run it with the buttons or Control-enter or Shift-enter to run a cell.
- Edit/create new cells, run again. Repeat indefinitely.
- You have a visual history of what you have run, with code and results nicely interspersed. With certain languages such as Python, you can plots and other things embedded, so that it becomes a complete reproducible story.

JupyterLab is the next iteration of this and has many more features, making it closer to an IDE or RStudio.

Notebooks are without a doubt a great tool. However, they are only one tool, and you need to know their limitations. See our other page on limitations of notebooks.

JupyterHub

Note: JupyterHub on Triton is still under development, and features will be added as they are needed or requested. Please use the Triton issue tracker.

The easiest way of using Jupyter is through JupyterHub - it is a multi-user jupyter server which takes a web-based login and spawns your own single-user server. This is available on Triton.

Connecting and starting

Currently jupyterHub is available only within Aalto networks, or from the rest of the internet after a first Aalto login: https://jupyter.triton.aalto.fi.

Once you log in, you must start your single-user server. There are several options available that trade off between long run time and short run time but more memory available. Your server runs in the Slurm queue, so the first start-up takes a few seconds but after that it will stay running even if you log out. The resources you request are managed by slurm: if you go over the memory limit, your server will be killed without warning or notification (but you can see it in the output log, ~/.jupyterhub_slurmspawner_* .log). The Jupyter server nodes are oversubscribed, which means that we can allocate more memory and CPU than is actually available. We will monitor the nodes to try to...
ensure that there are enough resources available, so do report problems to us. **Please request the minimum amount of memory you think you need** - you can always restart with more memory. You can go over your memory request a little bit before you get problems.

When you use Jupyter via this interface, the slurm billing weights are lower, so that the rest of your Triton priority does not decrease by as much.

**Usage**

Once you get to your single-user server Jupyter running as your own user on Triton. You begin in a convenience directory which has links to home, scratch, etc. You can not make files in this directory (it is read-only), but you can navigate to the other folders to create your notebooks. You have access to all the Triton filesystems (not project/archive) and all normal software.

We have some basic extensions installed:

- Jupyterlab (to use it, change /tree in the URL to /lab). Jupyterlab will eventually be made the default.
- modules integration
- jupyter_contrib_nbextensions - check out the variable inspector
- diff and merge tools (currently does not work somehow)

The log files for your single-user servers can be found in, see ~/jupyterhub_slurmspawner_.log. When a new server starts, these are automatically cleaned up when they are one week old.

For reasons of web security, you can’t install your own extensions (but you can install your own kernels). Send your requests to us instead.

**Problems? Requests?**

This service is currently in beta and under active development. If you notice problems or would like any more extensions or features, let us know. If this is useful to you, please let us know your user store, too. In the current development stage, the threshold for feedback should be very low.

Currently, the service level is best effort. The service may go down at any time and/or notebooks may be killed whenever there is a shortage of resources or need of maintenance. However, notebooks auto-save and do survive service restarts, and we will try to avoid killing things unnecessarily.

**Software and kernels**

We have various kernels automatically installed (these instructions should apply to both JupyterHub and sjupyter):

- Python (2 and 3 via some recent anaconda modules + a few more Python modules.)
- Matlab (latest module)
- Bash kernel
- R (a default R environment you can get by module load r-triton. (“R (safe)” is similar but tries to block some local user configuration which sometimes breaks things, see FAQ for more hints.)
- Julia: currently doesn’t seem to play nicely with global installations (if anyone knows something otherwise, let us know). Just load two modules: module load julia, module load jupyterhub/live, and then install the kernel julia and Pkg.add("IJulia") (and if it doesn’t enable, then using IJulia and installkernel("julia") to force re-installation) and it will install locally for you.
• We do not yet have a kernel management policy. Kernels may be added or removed over time. We would like to keep them synced with the most common Triton modules, but it will take some time to get this automatic. Send requests and problem reports.

Since these are the normal Triton modules, you can submit installation requests for software in these so that it is automatically available.

**Installing kernels from virtualenvs or Anaconda environments**

You can do this:

```bash
module load jupyterhub/live
envkernel conda --user --name INTERNAL_NAME --display-name="My conda" /path/to/conda_env
```

Or for Python virtualenvs:

```bash
module load jupyterhub/live
envkernel virtualenv --user --name INTERNAL_NAME --display-name="My virtualenv" /path/to/conda_env
```

You have to have the package `ipykernel` installed in the environment: activate the environment and do `pip install ipykernel`.

**Installing a different R module as a kernel**

Load your R modules, install R kernel normally (to some `NAME`), use envkernel as a wrapper to re-write the kernel (reading the `NAME` and rewriting to the same `NAME`), after it loads the modules you need:

```bash
# Load jupyterhub/live, and R 3.6.1 with IRkernel.
module load r-irkernel/1.1-python3
module load jupyterhub/live

# Use Rscript to install jupyter kernel
Rscript -e "library(IRkernel); IRkernel::installspec(name='NAME', displayname='R 3.6.1')"

# Use envkernel to re-write, loading the R modules.
envkernel lmod --user --kernel-template=NAME --name=NAME r-irkernel/1.1-python3
```

**Install your own kernels from other Python modules**

This works if the module has Python installed and working. This has to be done once in any Triton shell:

```bash
module load jupyterhub/live
envkernel lmod --user --name INTERNAL_NAME --display-name="Singularity my kernel" --MODULE_NAME
module purge
```
Install your own kernels from Singularity image

First, find the .simg file name. If you are using this from one of the Triton modules, you can use module show 
MODULE_NAME and look for SING_IMAGE in the output.

Then, install a kernel for your own user using envkernel. This has to be done once in any Triton shell:

```bash
module load jupyterhub/live
evtkernel singularity --user --name KERNEL_NAME --display-name="Singularity my kernel
˓→" SIMG_IMAGE
module purge
```

Install your own non-Python kernels

- First, module load jupyterhub/live. This loads the anaconda environment which contains all the 
  server code and configuration. (This step may not be needed for all kernels)
- Follow the instructions you find for your kernel. You may need to specify --user or some such to have it 
  install in your user directory.
- You can check your own kernels in ~/.local/share/jupyter/kernels/.

If your kernel involves loading a module, you can either a) load the modules within the notebook server (“soft-
wares” tab in the menu), or b) update your kernel.json to include the required environment variables (see kernel-
spec). (We need to do some work to figure out just how this works). Check /share/apps/jupyterhub/live/
miniconda/share/jupyter/kernels/ir/kernel.json for an example of a kernel that loads a module 
first.

Git integration

You can enable git integration on Triton by using the following lines from inside a git repository. (This is normal 
nbdiem, but uses the centrally installed one so that you don’t have to load a particular conda environment first. The 
```sed` command fixes relative paths to absolute paths, so that you use the tools no matter what modules you have 
loaded):

```bash
/share/apps/jupyterhub/live/miniconda/bin/nbdime config-git --enable
sed --in-place -r '@a=(\= )[ a-z/-]*\(git-nb\)@\1/share/apps/jupyterhub/live/miniconda/
˓→bin/\2' .git/config
```

FAQ/common problems

- Jupyterhub won’t spawn my server: “Error: HTTP 500: Internal Server Error (Spawner failed 
to start [status=1]).” Is your home directory quota exceeded? If that’s not it, check the ~/
jupyterhub_slurmspawner__* logs then contact us.
- My server has died mysteriously. This may happen if resource usage becomes too much and exceed the 
  limits - Slurm will kill your notebook. You can check the ~/jupyterhub_slurmspawner__* log files for 
jupyterhub to be sure.
- My server seems inaccessible / I can’t get to the control panel to restart my server. Especially with 
  JupyterLab. In JupyterLab, use File→Hub Control Panel. If you can’t get there, you can change the URL 
to /hub/home.
• **My R kernel keeps dying.** Some people seem to have global R configuration, either in `.bashrc` or `.Renviron` or some such which globally, which even affects the R kernel here. Things we have seen: pre-loading modules in `.bashrc` which conflict with the kernel R module; changing `RLIBS` in `.Renviron`. You can either (temporarily or permanently) remove these changes, or you could install your own R kernel. If you install your own, it is up to you to maintain it (and remember that you installed it).

• “Spawner pending” when you try to start - this is hopefully fixed in issue #1534/#1533 in JupyterHub. Current recommendation: wait a bit and return to JupyterHub home page and see if the server has started. Don’t click the button twice!

**See also**

• https://jupyter.org
  – Online demos and live tutorial: https://jupyter.org/try (use the Python one)

• Jupyter basic tutorial: https://www.youtube.com/watch?v=HW29067qVWk (this is just the first link on youtube - there are many more too)

• More advanced tutorial: Data Science is Software (this is not just a Jupyter tutorial, but about the whole data science workflow using Jupyter. It is annoying long (2 hours), but *very* complete and could be considered good "required watching")

• *Pitfalls of Jupyter Notebooks*

• CSC has this service, too, however there is no long term storage yet so there is limited usefulness for research: https://notebooks.csc.fi/

Our configuration is available on Github. Theoretically, all the pieces are here but it is not yet documented well and not yet generalizable. The Ansible role is a good start but the jupyterhub config and setup is hackish.

• Ansible config role: https://github.com/AaltoSciComp/ansible-role-fgci-jupyterhub

• Configuration and automated conda environment setup: https://github.com/AaltoSciComp/triton-jupyterhub

**Keras**

**supportlevel**

**pagelastupdated** 2020-02-20

**maintainer**

Keras is a neural network library which runs on tensorflow (among other things).

**Basic usage**

Keras is available in the anaconda module and some other anaconda modules. Run `module spider anaconda` to list available modules.

You probably want to learn how to run in the *GPU queues*. The other information in the *tensorflow* page also applies, especially the `--constraint` options to restrict to the GPUs that have new enough features.
Example

```bash
srun --gres=gpu:1 --pty bash
module load anaconda
python3
>>> import keras
Using TensorFlow backend.
>>> keras.__version__
'2.2.4'
```

Lammps

- **supportlevel**: C
- **pagelastupdated**: 2014

http://lammps.sandia.gov/

Building LAMMPS as a library

```bash
cd src
# default g++ compilation with system g++
module load openmpi/1.8.1-gcc
make -f Makefile.lib serial
```

Using Mathematica on Triton

**Load Mathematica**

Mathematica is loaded through a module:

```bash
module load mathematica
```

See available versions with `module avail mathematica`.

You can test by running in text-based mode:

```bash
$ wolfram
```

**With graphical user interface**

To launch the graphical user interface (GUI), login to triton.aalto.fi with `-X`, i.e. X11 forwarding enabled.

```bash
ssh -X triton.aalto.fi
```

If you need to run computationally-intensive things with the GUI, use `sinteractive` to get an interactive shell on a node:

```bash
sinteractive --mem=10G --time=1:00
```

Either way, you start the GUI with mathematica:
Running batch scripts

Create a script file, say script.m. You can run this script and store the outputs in output.txt using:

```
math -noprompt -run '<<script.m' > output.txt
```

To put this in a batch script, simply look at the serial jobs tutorial. Here is one such example:

```
#!/bin/bash
#SBATCH --mem=5G
#SBATCH --time=2:00
module load mathematica
math -noprompt -run '<<script.m'
```

Common problems

**Activation** If you need to activate Mathematica when you first run it, we recommend that you launch it in GUI mode first, choose ‘Other ways to activate’ then “Connect to a network license server”, and paste lic-mathematica.aalto.fi. It *should* be automatically activated, though, if not file an issue and link this page.

See also

Various other references also apply here once you load the module and adapt them to Slurm:

- [https://wiki.hpcc.msu.edu/display/hpccdocs/Using+Mathematica+in+Batch+Mode](https://wiki.hpcc.msu.edu/display/hpccdocs/Using+Mathematica+in+Batch+Mode)

Admin notes

When installing new versions, put !lic-mathematica.aalto.fi into Configuration/Licensing/mathpass in the base directory

Matlab

**Video**

See an example in the Winter Kickstart 2021 course

This page will guide you through the serial computing with Matlab at Triton cluster. (Note (2017): We used to have the Matlab Distributed Computing Server (MDCS), but because of low use we no longer have a license. You can still run in parallel on one node, up to 20-28 cores depending on how new.)
Matlab configuration

Matlab writes session data, compiled code and additional toolboxes to ~/.matlab. This can quickly fill up your $HOME quota. To fix this we recommend that you replace the folder with a symlink that points to a directory in your working directory.

rsync -lrt ~/.matlab/ $WRKDIR/matlab-config/ && rm -r ~/.matlab
ln -sT $WRKDIR/matlab-config ~/.matlab
quotafix -gs --fix $WRKDIR/matlab-config

Interactive usage

Interactive usage is currently available via the sinteractive tool. Do not use the cluster front-end for doing heavy task. Only meant for submitting jobs/compiling. Using MDCS for sending jobs is ok.

ssh -X user@triton.aalto.fi
sinteractive
module load matlab
matlab &

Simple serial script

Running a single core Matlab job is easy through the slurm queue. A sample slurm script is provided underneath:

#!/bin/bash -l
#SBATCH --time=00:05:00
#SBATCH --mem=100M
#SBATCH -o serial_Matlab.out
module load matlab
n=3
m=2
srun matlab -nojvm -nosplash -r "serial_Matlab($n,$m) ; exit(0)"

The above script can then be saved as a file (e.g. matlab_test.slrm) and the job can be submitted with sbatch matlab_test.slrm. The actual calculation is done in serial_Matlab.m-file:

function C = serial_Matlab(n,m)
  try
    A=0:(n*m-1);
    A=reshape(A,[2,3]).'

    B=2:(n*m+1);
    B=reshape(B,[2,3]).'

    C=0.5*ones(n,n)
    C=A*(B.' ) + 2.0*C
  catch error
    disp(getReport(error))
    exit(1)
  end
end

Remember to always set exit into your slurm script so that the program quits once the function serial_Matlab has finished. Using a try-catch-statement will allow your job to finish in case of any error within the program. If you
don’t do this, Matlab will drop into interactive mode and do nothing while your cluster time wastes.

NOTE: Starting from version r2019a the launch options -r ...; exit(0) can be easily replaced with the
-batch option which automatically exits matlab at the end of the command that is passed (see here for details).
So the last command from the slurm script above for Matlab r2019a will look like:

```
srun matlab -nojvm -nosplash -batch "serial_Matlab($n,$m);"
```

### Multiple serial batchjobs

The most common way to utilize Matlab is to write a single .M-file that can be used to run tasks as a non-interactive
batch job. These jobs are then submitted as independent tasks and when the heavy part is done, the results are collected
for analysis. For these kinds of jobs the Slurm array jobs is the best choice; For more information on array jobs see
Array jobs in the Triton user guide.

Below you will find an example how-to prepare and run such type of jobs.

#### run.m file doing the actual calculation task

The file below calculates Sin-function in the interval 0-2*PI and stores the results into a file. The interval is divided
into blocks that are distributed over the nodes.

```matlab
function run(blockIndex,pointsPerBlock,totalBlocks)
% blockindex runs from 0..totalblocks-1
% range 0..2pi
length=2*pi;
% values to setup even spacing between given range
% and splitting the spacings to even number of points per block
totalPoints=pointsPerBlock*totalBlocks;
step=length/(totalPoints-1);
start=blockIndex*pointsPerBlock*step;
% do some calculations, store the result so arrays A and B
for index=0:pointsPerBlock-1
    i=index+1;
    x=start+index*step;
    y=sin(x);
    A(i)=x;
    B(i)=y;
end
% save the results based on the blockIndex to a file
filename=strcat('output-',int2str(blockIndex));
save( filename, 'A', 'B', 'blockIndex');
% display message to output (log) that we have reached this far.
disp(sprintf('SUCCESS blockIndex %d',blockIndex));
% exit as this is a batch-job
exit;
```

### Submission of 10 independent tasks

Below the run.m is executed as an array job with 10 array tasks, which will execute independently, potentially in
parallel if there are enough idle resources. Note that it is using play partition with 5min time limit.

matslurm.sh:

```
#!/bin/bash -l
#SBATCH --time=00:05:00
#SBATCH --mem=500M
#SBATCH --array=1-10
#SBATCH -o job-%A.out
```

(continues on next page)
Aalto scientific computing guide

(continued from previous page)

```bash
#SBATCH --array=0-9
module load matlab
matlab -nojvm -r "run($SLURM_ARRAY_TASK_ID,100,10); quit"
```

Submit the job with “sbatch matslurm.sh” (or whatever you called the batch job script above).

**Collecting the results**

Finally a wrapper script to read in the .mat files and plots you the Sine function calculated in parallel with 10 tasks:

```matlab
function collectResults(numberOfBlocks)
    X=[];
    Y=[];
    for index=0:numberOfBlocks-1
        % read the output from the jobs
        filename = strcat( 'output-', int2str( index ) );
        load( filename );
        % cattenate results to a single arrays
        X=cat(2,X,A);
        Y=cat(2,Y,B);
    end
    plot(X,Y,'b+:')
```

**Seeding the random number generator**

Note that by default MATLAB always initializes the random number generator with a constant value. Thus if you launch several matlab instances e.g. to calculate distinct ensembles, then you need to seed the random number generator such that it’s distinct for each instance. In order to do this, you can call the `rng()` function, passing the value of `$SLURM_ARRAY_TASK_ID` to it.

**Parallel Matlab with Matlab’s internal parallelization**

Matlab has internal parallelization that can be activated by requesting more than one cpu per task in the Slurm script and using the `matlab_multithread` to start the interpreter.

```bash
#!/bin/bash -l
#SBATCH --time=00:15:00
#SBATCH --cpus-per-task=4
#SBATCH --mem=2G
#SBATCH --output=int_parallel.out
module load matlab
srun time -p matlab_multithread -nojvm -nosplash -r "int_parallel() ; exit(0)"
```

An example function is provided in this script

```matlab
function int_parallel()
    try
        tic;
        A = rand(2000,2000);
        A = A + A.';
        B = pinv(A);
        max(max(B * A))
    toc
```

(continues on next page)
catch error
    disp('Error occurred');
    exit(0)
end

Parallel Matlab with parpool

Often one uses Matlab’s parallel pool for parallelization. When using `parpool` one needs to specify the number of workers. This number should match the number of CPUs requested. `parpool` uses JVM so when launching the interpreter one needs to use `-nodisplay` instead of `-nojvm`. Example Slurm script:

```bash
#!/bin/bash -l
#SBATCH --time=00:15:00
#SBATCH --cpus-per-task=4
#SBATCH --mem-per-cpu=2G
#SBATCH -o parpool_parallel.out

module load matlab

srun matlab_multithread -nodisplay -r "parpool_parallel($SLURM_CPUS_PER_TASK) ; exit(0)"
```

An example function is provided in this script

```matlab
function parpool_parallel(n)
    try
        c=parcluster();
tempname()
mkdir(t);
c.JobStorageLocation=t;
parpool(c,n);
    catch error
        getReport(error)
disp('Error occurred');
        exit(0)
    end
end
```

6.1. Triton cluster
Parallel matlab in exclusive mode

```bash
#!/bin/bash -l
#SBATCH --time=00:15:00
#SBATCH --exclusive
#SBATCH -o parallel_Matlab3.out
export OMP_NUM_THREADS=$(nproc)
module load matlab/r2017b
matlab_multithread -nosplash -r "parallel_Matlab3($OMP_NUM_THREADS) ; exit(0)"
```

parallel_Matlab3.m:

```matlab
function parallel_Matlab3(n)
    % Try-catch expression that quits the Matlab session if your code crashes
    try
        % Initialize the parallel pool
        c=parcluster();
        t=tempname()
        mkdir(t)
        c.JobStorageLocation=t;
        parpool(c,n);
        % The actual program calls from matlab's example.
        % The path for r2017b
        addpath(strcat(matlabroot, '/examples/distcomp/main'));
        % The path for r2016b
        addpath(strcat(matlabroot, '/examples/distcomp'));
        pctdemo_aux_parforbench(10000,100,n);
    catch error
        getReport(error)
        disp('Error occurred');
        exit(0)
    end
end
```

Hints for Condor users

The above example also works (even nicer way) for condor.

A wrapper script to execute matlab on the department workstation.

```bash
#!/bin/bash -l
# a wrapper to run Matlab with condor
block=$1
pointsPerBlock=10
totalBlocks=10
matlab -nojvm -r "run($block,$pointsPerBlock,$totalBlocks)"
```

Condor submission script

Condor actually contains ArrayJob functionality that makes the task easier.

```bash
## Condor submit description (script) file for my_program.exe.
## 1. Specify the [path and] name for the executable file...
Executable = run.sh
```
## 2. Specify Condor execution environment.
Universe = vanilla
notify = Error

## 3. Specify remote execution machines running Linux (required)...
Requirements = ((OpSys == "Linux") || (OpSysName == "Ubuntu"))

## 4. Define input files and arguments
#Input = stdin.txt.$(Process)
Arguments = $(Process)

## 5. Define output/error/log files
Output = log/stdout.$(Process).txt
Error = log/stderr.$(Process).txt
Log = log/log.$(Process).txt

## 6. Tell Condor which files need to be transferred and when.
Transfer_input_files = run.m
Transfer_output_files = output-$($(Process)).mat
Transfer_executable = true
Should_transfer_files = YES
When_to_transfer_output = ON_EXIT

## 7. Add 10 copies of the job to the queue
Queue 10

### FAQ / troubleshooting

If things randomly don’t work, you can try removing or moving either the ~/.matlab directory or ~/.matlab/Rxxxxxy directory to see if it’s caused by configuration.

Random error messages about things not loading and/or something (Matlab Live Editor maybe) doesn’t work: ls *.m, do you have any unexpected files like pathdef.m in there? Remove them.

Also, check your home quota. Often .matlab gets large and fills up your home directory. Check the answer at the very top of the page, under “Matlab Configuration”.

### MLPack

```
pagelastupdated 2014
supportlevel C
https://www.mlpack.org/

1. module load cmake; module load armadillo/4.3-mkl; module load mkl
2. mkdir build && cd build
3. cmake -D ARMADILLO_LIBRARY=$ARMADILLO_LIBRARY -D AR-
   MADILLO_INCLUDE_DIR=$ARMADILLO_INCLUDE ../
4. make
5. bin/mlpack_test
6. make install CMAKE_INSTALL_PREFIX=/share/apps/mlpack/1.0.8
```

For newer boost library also load boost module and tell cmake where to find boost
module load boost
...
cmake -D BOOST_ROOT=$BOOST_ROOT -D ARMADILLO_LIBRARY=$ARMADILLO_LIBRARY -D ARMADILLO_INCLUDE_DIR=$ARMADILLO_INCLUDE ../..

Notes

- 1.0.10 installation failed when installing doc to /usr/local (install prefix defined as /share/apps/mlpack/1.0.10). The solution was manually tuned install prefix at cmake_install.cmake

MNE

module load mne

Follow the instruction to source the init script specific to your shell. In the directory:

$MNE_ROOT/..

you can find the release notes, the manual, and some sample data.

We do not recommend using the MNE command line tools, a more modern solution is to use the MNE-python suite.

NVIDIA’s singularity containers

supportlevel A
pagelastupdate 2020-05-15
maintainer

NVIDIA provides many different docker images containing scientific software through their NGC repository. This software is available for free for NVIDIA's GPUs and one can register for free to get access to the images.

You can use these images as a starting point for your own GPU images, but do be mindful of NVIDIA's terms and conditions. If you want to store your own images that are based on NGC images, either use NGC itself or our own Docker registry that is documented on the singularity containers page.

We have converted some of these images with minimal changes into singularity images that are available in Triton.

Currently updated images are:

- **nvidia-tensorflow**: Contains tensorflow. Due to major changes that happened between Tensorflow v1 and v2, image versions have either tf1 or tf2 to designate the major version of Tensorflow.
- **nvidia-pytorch**: Contains PyTorch.

There are various other images available that can be installed very quickly if required.
Running simple Tensorflow/Keras model with NVIDIA’s containers

Let’s run the MNIST example from Tensorflow’s tutorials:

```python
model = tf.keras.models.Sequential(
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
)
```

The full code for the example is in `tensorflow_mnist.py`. One can run this example with `srun`:

```
wget https://raw.githubusercontent.com/AaltoSciComp/scicomp-docs/master/triton/examples/tensorflow/tensorflow_mnist.py
module load nvidia-tensorflow/20.02-tf1-py3
srun --time=00:15:00 --gres=gpu:1 singularity_wrapper exec python tensorflow_mnist.py
```

or with `sbatch` by submitting `tensorflow_singularity_mnist.sh`:

```bash
#!/bin/bash
#SBATCH --gres=gpu:1
#SBATCH --time=00:15:00
module load nvidia-tensorflow/20.02-tf1-py3
singularity_wrapper exec python tensorflow_mnist.py
```

Do note that by default Keras downloads datasets to `$HOME/.keras/datasets`.

Running simple PyTorch model with NVIDIA’s containers

Let’s run the MNIST example from PyTorch’s tutorials:

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

The full code for the example is in `pytorch_mnist.py`. One can run this example with `srun`:
wget https://raw.githubusercontent.com/AaltoSciComp/scicomp-docs/master/triton/examples/pytorch/pytorch_mnist.py
module load nvidia-pytorch/20.02-py3
srun --time=00:15:00 --gres=gpu:1 singularity_wrapper exec python pytorch_mnist.py

or with sbatch by submitting pytorch_singularity_mnist.sh:

```bash
#!/bin/bash
#SBATCH --gres=gpu:1
#SBATCH --time=00:15:00
module load nvidia-pytorch/20.02-py3
singularity_wrapper exec python pytorch_mnist.py
```

The Python-script will download the MNIST dataset to data folder.

**Octave**

From Octave's web page: GNU Octave is a high-level language, primarily intended for numerical computations. It provides a convenient command line interface for solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with Matlab. It may also be used as a batch-oriented language.

Octave has extensive tools for solving common numerical linear algebra problems, finding the roots of nonlinear equations, integrating ordinary functions, manipulating polynomials, and integrating ordinary differential and differential-algebraic equations. It is easily extensible and customizable via user-defined functions written in Octave’s own language, or using dynamically loaded modules written in C++, C, Fortran, or other languages.

**Getting started**

Simply load the latest version of Octave.

```bash
module load octave
octave
```

It is best to pick a version of octave and stick with it. Do module spider octave and use the whole name:

```bash
module load octave/4.4.1-qt-python2
```

To run octave with the GUI, run it with:

```bash
octave --force-gui
```
Installing packages

Before installing packages you should create a file ~/.octaverc with the following content:

```octave
package_dir = ['/scratch/work/', getenv('USER'), '/octave'];
eval(['pkg prefix ', package_dir, ';']);
setenv("CXX","g++ -std=gnu++11")
setenv("DL_LD","g++ -std=gnu++11")
setenv("LD_CXX","g++ -std=gnu++11")
setenv("CC","gcc")
setenv("F77","gfortran")
```

This sets up /scratch/work/$USER/octave to be your Octave package directory and sets gcc to be your compiler. By setting Octave package directory to your work directory you won’t run into any quota issues.

After this you should load gcc- and texinfo-modules. This gives you an up-to-date compiler and tools that Octave uses for its documentation:

```octave
module load gcc
module load texinfo
```

Now you can install packages in octave with e.g.:

```octave
pkg install -forge -local io
```

After this you can unload the gcc- and texinfo-modules:

```octave
module unload gcc
module unload texinfo
```

OpenFOAM (with ParaView)

This uses Singularity containers, so you should refer to that page first for general information.

OpenFOAM and ParaView have been installed from the Ubuntu 16.04 Docker image provided by OpenFOAM people. It has minimal amount of other software installed.

Parallelization is done against Triton’s OpenMPI, so using this container with other OpenMPI modules is discouraged.

New image (singularity-openfoam)

Loading: simply module load singularity-openfoam and use singularity_wrapper.

OpenFOAM is installed in /opt/OpenFOAM. The OpenFOAM bashrc file is automatically sourced when you exec or shell within the image to set PATH and so on.
**Old image (OpenFOAM)**

This is quite similar to the new image.

Within the container OpenFOAM is installed under `/opt/openfoam4/` and ParaView under `/opt/paraviewopenfoam50/`. PATH is automatically appended with their respective paths so all program calls are available automatically.

**Usage**

(This has not been updated to the new image yet. To change to new image, don’t do the `module use` and instead just load `singularity-openfoam`.)

This example shows how you can run damBreak example. Firstly, let’s load the OpenFOAM module and create a folder for the example:

```
module use /share/apps2/singularity/modules
module load OpenFOAM
mkdir damBreak
cd damBreak
```

Secondly, let’s use singularity shell to copy example data files to the folder and to initialize the simulation:

```
cp -r /opt/openfoam4/tutorials/multiphase/interFoam/laminar/damBreak/damBreak/0 .
cp -r /opt/openfoam4/tutorials/multiphase/interFoam/laminar/damBreak/damBreak/constant .
blockMesh
decomposePar
exit
```

After this one can submit the following slurm script with `sbatch` to solve the problem:

```
#!/bin/bash
#SBATCH --time=00:30:00
#SBATCH --ntasks=4
#SBATCH --mem=4G
module use /share/apps2/singularity/modules
module purge
module load OpenFOAM
srun singularity_wrapper exec interFoam -parallel
```

Paraview can be started similarly with this script:

```
#!/bin/bash
#SBATCH --time=00:10:00
#SBATCH --mem=8G
module use /share/apps2/singularity/modules
module purge
module load OpenFOAM
singularity_wrapper exec paraview
```
OpenPose

This uses *Singularity containers*, so you should refer to that page first for general information.

OpenPose has been compiled against OpenBlas, Caffe, CUDA and cuDNN. Image is based on a `nvidia/cuda:10.1-cudnn7-devel-ubuntu18.04` docker image.

Dockerfile for this image is available [here](#).

Within the container OpenPose is installed under `/opt/openpose`. Due to the way the libraries are organized, `singularity_wrapper` changes the working directory to `/opt/openpose`.

Running OpenPose example

One can run this example with `srun`:

```bash
module load singularity-openpose
sbatch openpose.slrm
```

Example `sbatch` script is shown below.

```bash
#!/bin/bash
#SBATCH --time=00:10:00
#SBATCH --mem=8G
#SBATCH --gres=gpu:1
module load singularity-openpose/v1.5.1
# Print out usage flags
singularity_wrapper exec openpose --help
# Run example
singularity_wrapper exec openpose --video /opt/openpose/examples/media/video.avi --
--display 0 --write_video $(pwd)/openpose.avi
```

Paraview

As a module

A serial version is available on login2. You will need to use the “forward connection” strategy by using `ssh` port forwarding. For example, `run ssh -L BBBB:nnnNNN:AAAA username@triton`, where BBBB is the server you connect to locally and nnnNNN is the node name and AAAA is the port on that node. See [this FAQ question](#).

See issue #13: [https://version.aalto.fi/gitlab/AaltoScienceIT/triton/issues/13](#) for some user experiences. (Note: the author of this entry is not a paraview expert, suggestions welcome.)
As a container

You can also use paraview via *Singularity containers*, so you should refer to that page first for general information. It is part of the *OpenFOAM (with ParaView)* container.

Python

See an example in the Winter Kickstart 2021 course

Python is widely used programming language where we have installed all basic packages on every node. Yet, python develops quite fast and the system provided packages are often not complete or getting old.

Python distributions

<table>
<thead>
<tr>
<th>I don’t really care, I just want recent stuff and to not worry.</th>
<th>Python to use</th>
<th>How to install own packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anaconda</td>
<td>module load</td>
<td></td>
</tr>
<tr>
<td>anaconda</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simple programs with common packages, not switching between Pythons often</td>
<td>Anaconda</td>
<td>pip install</td>
</tr>
<tr>
<td>anaconda</td>
<td>--user</td>
<td></td>
</tr>
<tr>
<td>Your own conda environment</td>
<td>Miniconda</td>
<td>conda environment</td>
</tr>
<tr>
<td>miniconda</td>
<td>+ conda</td>
<td></td>
</tr>
<tr>
<td>Your own virtual environment</td>
<td>Module virtualenv</td>
<td>virtualenv + pip + setuptools</td>
</tr>
<tr>
<td>py-virtualenv</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The main version of modern Python is 3. Support for old Python 2 ended at the end of 2019. There are also different distributions: The “regular” CPython, Anaconda (a package containing CPython + a lot of other scientific software all bundled together), PyPy (a just-in-time compiler, which can be much faster for some use cases). Triton supports all of these.

- For general scientific/data science use, we suggest that you use Anaconda. It comes with the most common scientific software included, and is reasonably optimized.

- There are many other “regular” CPython versions in the module system. These are compiled and optimized for Triton, and are highly recommended. The default system Python is old and won’t be updated.

Make sure your environments are **reproducible** - you can recreate them from scratch. History shows you will probably have to do this eventually, and it also ensures that others can always use your code. We recommend a minimal [requirements.txt](pip) or [environment.yml](conda), hand-created with the minimal dependencies in there.
Quickstart

Use `module load anaconda` (or `module load anaconda2` for Python 2) to get a modern Python.

If you have simple needs, use `pip install --user` to install packages. For complex needs, use `anaconda + conda environments` to isolate your projects.

Install your own packages easily

Installing your own packages with `pip install` won’t work, since it tries to install globally for all users. Instead, you should do this (add `--user`) to install the package in your home directory (`~/.local/lib/pythonN.N/`):

```
pip install --user $package_name
```

This is quick and effective best used for leaf packages without many dependencies and if you don’t switch Python modules often.

**Warning:** `pip install --user` can result in incompatibilities

If you do this, then the module will be shared among all your projects. It is quite likely that eventually, you will get some incompatibilities between the Python you are using and the modules installed. In that case, you are on your own (simple recommendation is to remove all modules from `~/.local/lib/pythonN.N` and reinstall).

**If you get incompatible module errors, our first recommendation will be to remove everything installed this way and use conda/virtual environments instead.** It’s not a bad idea to do this when you switch to environments anyway.

If you encounter problems, remove all your user packages:

```
rm -r ~/.local/lib/python*.*/
```

and reinstall everything after loading the environment you want.

**Note:** Example of dangers of `pip install --user`

Someone did `pip install --user tensorflow`. Some time later, they noticed that they couldn’t use Tensorflow + GPUs. We couldn’t reproduce the problem, but in the end found they had this local install that was hiding any Tensorflow in any module (forcing a CPU version on them).

**Note:** `pip` installs from the Python Package Index.

Anaconda and conda environments

`Anaconda` is a Python distribution by Continuum Analytics (open source, of course). It is nothing fancy, they just take a lot of useful scientific packages and their dependencies and put them all together, make sure they work, and do some optimization. They also include most of the most common computing and data science packages and non-Python compiled software and libraries. It is also all open source, and is packaged nicely so that it can easily be installed on any major OS.

To load anaconda, use the module system (you can also load specific versions):

```
module load anaconda  # python3
module load anaconda2 # python2
```
**Note:** Before 2020, Python3 was via the anaconda3 module (note the 3 on the end). That’s still there, but in 2020 we completely revised our Anaconda installation system, and dropped active maintenance of Python 2. All updates are in anaconda only in the future.

**Conda environments**

If you encounter a situation where you need to create your own environment, we recommend that you use conda environments. When you create your own environment the packages from the base environment (default environment installed by us) will not be used, but you can choose which packages you want to install.

We nowadays recommend that you use the miniconda-module for installing these environments. Miniconda is basically a minimal Anaconda installation that can be used to create your own environments.

Do note that these environments can be quite big and if you have multiple environments installed you can run into quota issues in your /home. If you encounter such issues do contact us. In the past we recommended installing conda environments under $WRKDIR, but this can cause file system problems when launching array jobs and is thus no longer recommended.

**virtualenv** does not work with Anaconda, use conda instead.

- Load the miniconda module. You should look up the version and use load same version each time you source the environment:

  ```bash
  # Load miniconda first. This must always be done before activating the env!
  module load miniconda
  ```

- Create an environment. This needs to be done once:

  ```bash
  # create environment with the packages you require
  conda create -n ENV_NAME python pip ipython tensorflow-gpu pandas ...
  ```

- Activate the environment. This needs to be done every time you load the environment:

  ```bash
  # This must be run in each shell to set up the environment variables properly.
  # make sure module is loaded first.
  source activate ENV_NAME
  ```

- Activating and using the environment, installing more packages, etc. can be done either using conda install or pip install:

  ```bash
  # Install more packages, either conda or pip
  conda search PACKAGE_NAME
  conda install PACKAGE_NAME
  pip install PACKAGE_NAME
  ```

- Leaving the environment when done (optional):

  ```bash
  # Deactivate the environment
  source deactivate
  ```

- Worst case, you have incompatibility problems. Remove everything, including the stuff installed with pip install --user. If you’ve mixed your personal stuff in with this, then you will have to separate it out:

  ```bash
  # Remove anything installed with pip install --user.
  rm -r ~/.local/lib/python*.*/
  ```
A few notes about conda environments:

- Once you use a conda environment, everything goes into it. Don’t mix versions with, for example, local packages in your home dir and `--pip install --user`. Things installed (even previously) with `pip install --user` will be visible in the conda environment and can make your life hard! Eventually you’ll get dependency problems.
- Often the same goes for other python based modules. We have setup many modules that do use anaconda as a backend. So, if you know what you are doing this might work.

**Python: virtualenv**

Virtualenv is default-Python way of making environments, but does **not** work with Anaconda. We generally recommend using anaconda, since it includes a lot more stuff by default, but virtualenv works on other systems easily so it’s good to know about.

```bash
# Load module python
module load py-virtualenv

# Create environment
virtualenv DIR

# activate it (in each shell that uses it)
source DIR/bin/activate

# install more things (e.g. ipython, etc.)
pip install PACKAGE_NAME

# deactivate the virtualenv
deactivate
```

**IPython Parallel**

`ipyparallel` is a tool for running embarrassingly parallel code using Python. The basic idea is that you have a *controller* and *engines*. You have a *client* process which is actually running your own code.

Preliminary notes: `ipyparallel` is installed in the anaconda{2,3}/latest modules.

Let’s say that you are doing some basic interactive work:

- **Controller**: this can run on the frontend node, or you can put it on a script. To start: `ipcontroller --ip="*"`
- **Engines**: `srun -N4 ipengine`: This runs the four engines in slurm interactively. You don’t need to interact with this once it is running, but remember to stop the process once it is done because it is using resources. You can start/stop this as needed.
- **Start your Python process and use things like normal:**

```python
import os
import ipyparallel
client = ipyparallel.Client()
result = client[:].apply_async(os.getpid)
pid_map = result.get_dict()
print(pid_map)
```
This method lets you turn on/off the engines as needed. This isn’t the most advanced way to use ipyparallel, but works for interactive use.

See also: IPython parallel for a version which goes in a slurm script.

**Background: pip vs python vs anaconda vs conda vs virtualenv**

Virtual environments are self-contained python environments with all of their own modules, separate from the system packages. They are great for research where you need to be agile and install whatever versions and packages you need.

We highly recommend virtual environments or conda environments (below)

- Anaconda: use conda, see below
- Normal Python: virtualenv + pip install, see below

You often need to install your own packages. Python has its own package manager system that can do this for you. There are three important related concepts:

- **pip**: the Python package installer. Installs Python packages globally, in a user’s directory (`--user`), or anywhere. Installs from the Python Package Index.
- **virtualenv**: Creates a directory that has all self-contained packages that is manageable by the user themself. When the virtualenv is activated, all the operating-system global packages are no longer used. Instead, you install only the packages you want. This is important if you need to install specific versions of software, and also provides isolation from the rest of the system (so that you work can be uninterrupted). It also allows different projects to have different versions of things installed. virtualenv isn’t magic, it could almost be seen as just manipulating `PYTHONPATH`, `PATH`, and the like. Docs: https://docs.python-guide.org/dev/virtualenvs/
- **conda**: Sort of a combination of package manager and virtual environment. However, it only installed packages into environments, and is not limited to Python packages. It can also install other libraries (C, fortran, etc) into the environment. This is extremely useful for scientific computing, and the reason it was created. Docs for envs: https://conda.io/projects/conda/en/latest/user-guide/concepts/environments.html.

So, to install packages, there is **pip** and **conda**. To make virtual environments, there is **venv** and **conda**.

Advanced users can see this rosetta stone for reference.

On Triton we have added some packages on top of the Anaconda installation, so cloning the entire Anaconda environment to local conda environment will not work (not a good idea in the first place but some users try this every now and then).

**Examples**

**Running Python with internal parallelization (OpenMP)**

A simple parallel Python script using OpenMP. Both anaconda modules and optimized Python modules support OpenMP, but optimized versions are faster.
**Python OpenMP example**

parallel_Python.slrm:

```bash
#!/bin/bash
#SBATCH --time=00:10:00
#SBATCH --cpus-per-task=4
#SBATCH --mem=2G
#SBATCH -o parallel_Python.out

module load anaconda

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

srun -c $SLURM_CPUS_PER_TASK python parallel_Python.py
```

parallel\Python.py:

```python
import numpy as np
a = np.random.random([2000,2000])
a = a + a.T
b = np.linalg.pinv(a)
print(np.amax(np.dot(a,b))
```

**Running MPI parallelized Python with mpi4py**

MPI parallelized Python requires a valid MPI installation that support our SLURM scheduler. Thus anaconda is not the best option. We have installed MPI-supporting Python versions to different toolchains.

Using mpi4py is quite easy. Example is provided below.

**Python MPI4py**

A simple script mpi4py.py that utilizes mpi4py.

```python
#!/usr/bin/env python

"""
Parallel Hello World
"""

from mpi4py import MPI
import sys
size = MPI.COMM_WORLD.Get_size()
rank = MPI.COMM_WORLD.Get_rank()
name = MPI.Get_processor_name()
sys.stdout.write("Hello, World! I am process \$d of \$d on \$s.\n"
 % (rank, size, name))
```

Running mpi4py.py using only srun:

```
module load Python/2.7.11-goolf-triton-2016b
srun --time=00:10:00 --ntasks=4 -p debug python helloworld.py
```

Example sbatch script mpi4py.slrm when running mpi4py.py through sbatch:
PyTorch

supportlevel A

pagelastupdated 2020-05-15

maintainer

PyTorch is a commonly used Python package for deep learning.

Basic usage

First, check the tutorials up to and including GPU computing.

If you plan on using NVIDIA’s containers to run your model, please check the page about NVIDIA’s singularity containers.

The basic way to use is via the Python in the anaconda module. If you’re not using Tensorflow as well, you can pick either -tf1- or -tf2-version. If you’re using Tensorflow as well, please check our Tensorflow page.

Don’t load any additional CUDA modules, anaconda includes everything.

If you use GPUs, you need --constraint='kepler|pascal|volta' in order to select a GPU new enough to run tensorflow. (Note that as we get newer cards, this will need further updating).

Simple PyTorch model

Let’s run the MNIST example from PyTorch’s tutorials:

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fcl = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

The full code for the example is in tensorflow_mnist.py. One can run this example with srun:
Running simple PyTorch model with NVIDIA’s containers

Let’s run the MNIST example from PyTorch’s tutorials:

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

The full code for the example is in `pytorch_mnist.py`. One can run this example with `srun`:

```bash
wget https://raw.githubusercontent.com/AaltoSciComp/scicomp-docs/master/triton/
--examples/pytorch/pytorch_mnist.py
module load nvidia-pytorch/20.02-\py3
srun --time=00:15:00 --gres=gpu:1 singularity_wrapper exec python pytorch_mnist.py
```

or with `sbatch` by submitting `pytorch_singularity_mnist.sh`:

```bash
#!/bin/bash
#SBATCH --gres=gpu:1
#SBATCH --time=00:15:00
module load nvidia-pytorch/20.02-py3
singularity_wrapper exec python pytorch_mnist.py
```

The Python-script will download the MNIST dataset to `data` folder.
Common problems

- Random CUDA errors: don’t load any other CUDA modules, only anaconda. Anaconda includes the necessary libraries in compatible versions.

R

Video

See an example in the Winter Kickstart 2021 course

R is a language and environment for statistical computing and graphics with wide userbase. There exists several packages that are easily imported to R.

Getting started

Simply load the latest R.

```
module load r
```

As any packages you install against R are specific to the version you installed them with, it is best to pick a version of R and stick with it. You can do this by checking the R version with `module spider r` and using the whole name when loading the module:

```
module load r/3.6.1-python3
```

If you want to detect the number of cores, you should use the proper Slurm environment variables (defaulting to all cores):

```
library(parallel)
as.integer(Sys.getenv('SLURM_CPUS_PER_TASK', parallel:::detectCores()))
```

Installing packages

There are two ways to install packages.

1. You can usually install packages yourself, which allows you to keep up to date and reinstall as needed. Good instructions can be found here, for example:

```
R
> install.packages('L1pack')
```

This should guide you to selecting a download mirror and offer you the option to install in your home directory. If you have a lot of packages, you can run out of home quota. In this case you should move your package directory to your work directory and replace it the `~/R`-directory with a symlink that points to your `$WRKDIR/R`.

Example of doing this is here:
mv ~/R $WRKDIR/R
ln -s $WRKDIR/R ~/R

More info on R library paths can be found here. Looking at R startup can also be informative.

2. You can also put a request to the triton issue tracker and mention which R-version you are using.

Simple R serial job

Serial R example

r_serial.slrm:

#!/bin/bash -l
#SBATCH --time=00:05:00
#SBATCH --ntasks=1
#SBATCH --mem=100M
#SBATCH --output=r_serial.out

module load r
n=3
m=2
srun Rscript --vanilla r_serial.R $n $m

r_serial.R:

args = commandArgs(trailingOnly=TRUE)

n<-as.numeric(args[1])
m<-as.numeric(args[2])

print(n)
print(m)

A<-t(matrix(0:5,ncol=n,nrow=m))
print(A)
B<-t(matrix(2:7,ncol=n,nrow=m))
print(B)
C<-matrix(0.5,ncol=n,nrow=n)
print(C)

C<-A %*% t(B) + 2*C
print(C)

Simple R job using OpenMP for parallelization

R OpenMP Example

r_openmp.slrm:

#!/bin/bash
#SBATCH --time=00:15:00
#SBATCH --cpus-per-task=4

(continues on next page)
 module load r
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
time srun Rscript --default-packages=methods,utils,stats R-benchmark-25.R

The benchmark script is available [here](#) (more information about it is available [here](#) page).

**Simple R parallel job using 'parallel'-package**

**Parallel R example**

`r_parallel.slrm`

```bash
#!/bin/bash
#SBATCH --time=00:20:00
#SBATCH --cpus-per-task=4
#SBATCH --mem=2G
#SBATCH --output=r_parallel.out

# Set the number of OpenMP-threads to 1,
# as we're using parallel for parallelization
export OMP_NUM_THREADS=1

# Load the version of R you want to use
module load r

# Run your R script
srun Rscript r_parallel.R
```

`r_parallel.R`

```r
library(pracma)
library(parallel)
invertRandom <- function(index) {
  A<-matrix(runif(2000*2000),ncol=2000,nrow=2000);
  A<-A + t(A);
  B<-pinv(A);
  return(max(B %*% A));
}
ptm<-proc.time()
mclapply(1:16,invertRandom, mc.cores=Sys.getenv('SLURM_CPUS_PER_TASK'))
proc.time()-ptm
```

When constrained to opt-architecture, run times for different core numbers were

<table>
<thead>
<tr>
<th>ncores</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>runtime</td>
<td>380.757</td>
<td>182.185</td>
<td>125.526</td>
<td>84.230</td>
</tr>
</tbody>
</table>
RStan

**supportlevel** B

**pagelastupdated** 2018-07-26

**maintainer**

RStan is an R interface to Stan. Stan is a platform for modeling.

**Basic installation**

RStan is installed as an R package and there is nothing too special about it.

First, load the R module you need to use. There are different options, using different compilers. Do **not** use aniomkl R version, because it requires the intel compilers to work on the nodes to compile every time you run, and they aren’t available there. If you load a **goolf** R version, it will work (you could work around this by pre-compiling models, if you wanted):

```
$ module spider R
...
R/3.4.1-goolf-triton-2017a
R/3.4.1-iomkl-triton-2017a

$ module load R/3.4.1-goolf-triton-2017a
```

If you change R versions (from intel to gcc) or get errors about loading libraries, you may have installed incompatible libraries. Removing your ~/.R directory and reinstalling all of your libraries is a good first place to start.

**Notes**

You should detect the number of cores with:

```
as.integer(Sys.getenv('SLURM_JOB_CPUS_PER_NODE', parallel::detectCores()))
```

**Common Rstan problems**

- Models must be compiled on the machine that is running them, Triton or other workstations. The compiled model files aren’t necessarily portable, since they depend on the libraries available when build. One symptom of this problem is error messages which talk about loading libraries and GLIBC_2.23 or some such.

- In order to compile models, you must have the compiler available on the nodes. Thus, the Intel compilers (**iomkl**) won’t work. It also won’t work if the Intel compiler license servers are down. Using the GNU compiler toolchains are more reliable.
Example

RStudio

<table>
<thead>
<tr>
<th>supportlevel</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>pagelastupdated</td>
<td>2014</td>
</tr>
</tbody>
</table>

https://www.rstudio.com/ is an IDE for R

```
module load R/3.1.1-openblas boost/1.56 cmake/2.8.12.2 gcc/4.9.1 PrgEnv-gnu/0.1 qt/4.8.6
mkdir build && cd build
make .. -DRSTUDIO_TARGET=Desktop -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/share/apps/rstudio/0.98/ -DBOOST_ROOT=$BOOST_ROOT
```

Siesta & Transiesta

Copy-pasted Makefiles from Rocks. Should be used as a starting point. If you have a fully working version for SL6.2, send us a copy please.

See old wiki: https://wiki.aalto.fi/display/Triton/Applications

Rename siesta-3.0.arch.make.xxx => siesta-3.0-b/Obj/arch.make

Your own notebooks via sjupyter

**Note:** Now that Triton Jupyterhub exists, this method of running Jupyter is not so important. It is only needed if you need more resources than JupyterHub can provide.

We provide a command sjupyter which automates launching your own notebooks in the Slurm queue. To use this, module load sjupyter. This gives you more flexibility in choosing your nodes and resources than Jupyterhub, but also will after your and your department’s Triton priority more because you are blocking others from using these resources.

Set up the proxy

When running Jupyter on another system, the biggest problem is always making the connection securely. To do this here, we use a browser extension and SSH Proxy.

- Install the proxy extension
  - Install the extension FoxyProxy Standard (Firefox or Chrome). Some versions do not work properly: the 5.x series for Firefox may not work, but older and newer does.
- Create a new proxy rule with the pattern *.int.triton.aalto.fi* (or jupyter.triton.aalto.fi if you want to connect to that using the proxy).
  - Proxy type: SOCKS5, Proxy URL: localhost, port 8123.
  - DNS through the proxy: on.
• SSH to triton and use the \texttt{-D 8123}. This starts a proxy on your computer on port 8123. This has to always be running whenever you connect to the notebook.

  – If you are in Aalto networks: \texttt{ssh -D 8123 username@triton.aalto.fi}.

  – If you are not in Aalto networks, you need to do an extra hop through another Aalto server: \texttt{ssh -D 8123 username@triton.aalto.fi -o ProxyCommand='ssh username@kosh.aalto.fi -W %h:%p'}.

Now, when you go to any address matching *.*.int.triton.aalto.fi*, you will automatically connect to the right place on Triton. You can use Jupyter like normal. But if the ssh connection goes down, then you can’t connect and will get errors, so be aware (especially with jupyter.triton.aalto.fi which you might expect to always work).

**Starting sjupyter**

We have the custom-built command \texttt{sjupyter} for starting Jupyter on Triton.

First, you must load the \texttt{sjupyter} module:

```
module load sjupyter
```

To run in the Triton queue (using more resources), just use \texttt{sjupyter}. This will start a notebook on the interactive Slurm queue. All the normal rules apply: timelimits, memory limits, etc. If you want to request more resources, use the normal Slurm options such as \texttt{-t}, \texttt{--mem}, etc. Notebooks can only last as long as your job lasts, and you will need to restart them. Be efficient with resource usage: if you request a lot of resources and leave the notebook idle, no one else can use them. Thus, try to use the (default) interactive partition, which handles this automatically.

To run on the login node, run \texttt{sjupyter --local}. This is good for small testing and so on, which doesn’t use too much CPU or memory.

**Spyder**

```
supportlevel C
pagelastupdated 2014
```

Spyder is the Scientific P'Ython Development EnviRonment:https://www.spyder-ide.org/

This guide shows you how to set this up with different version of Qt4 and python compared to the default version provided by operating system. Virtual environment makes this encapsulated from the rest of the environment and thus you can install different versions of python packages and also make the environment more portable.

**Load pre-set environment modules**

```
module load triton/python/2.7.6
module load qt/4.8.6
```
Aalto scientific computing guide

Setup you virtualenv

```bash
mkdir -p /local/mhhakala/virtualenv && cd /local/mhhakala/virtualenv
virtualenv spyder_env
source spyder_env/bin/activate
```

Install SIP + PyQt to the virtualenv

```bash
# note, that we now have the virtualenv spyder_env activated
# SIP/PyQt4 do not install with pip, so download first to some location
# tar xzf sip-4.16.7.tar.gz
cd sip-4.16.7
python configure.py
make && make install

tar xzf PyQt-x11-gpl-4.10.4.tar.gz
cd PyQt-x11-gpl-4.10.4
python configure.py
make
make install
```

Install spyder to the virtualenv

```bash
# still under activated spyder_env
pip install spyder
```

Tensorflow

- supportlevel A
- pagelastupdated 2020-05-15
- maintainer

Tensorflow is a commonly used Python package for deep learning.

Basic usage

First, check the tutorials up to and including GPU computing.

If you plan on using NVIDIA's containers to run your model, please check the page about NVIDIA's singularity containers.

The basic way to use is via the Python in the anaconda module. The versions with -tf2 (the default ones) have Tensorflow 2 installed. If you use module spider anaconda, you can see a -tf1 version available.

**Warning:** Older versions of Tensorflow were CPU-only or GPU-only

With older versions of tensorflow (<1.15.0), you have to decide at install time if you want a version that runs on CPUs or GPUs. This means that we can’t install it for everyone and expect it to work everywhere - you have to
load something different if you want it to run on login node/regular nodes (probably for testing) or GPU nodes. The old \texttt{-cpu} and \texttt{-gpu} versions in the \texttt{anaconda2}- and \texttt{anaconda3}-modules denoted this.

From tensorflow versions $\geq 1.15.0$, they solved this problem (thankfully)

Don’t load any additional CUDA modules, anaconda includes everything.

If you use GPUs, you need \texttt{--constraint='kepler|pascal|volta'} in order to select a GPU new enough to run tensorflow. (Note that as we get newer cards, this will need further updating).

**Simple Tensorflow/Keras model**

Let’s run the MNIST example from Tensorflow’s tutorials:

```python
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

The full code for the example is in \texttt{tensorflow_mnist.py}. One can run this example with \texttt{srun}:

```bash
wget https://raw.githubusercontent.com/AaltoSciComp/scicomp-docs/master/triton/examples/tensorflow/tensorflow_mnist.py
module load anaconda
srun --time=00:15:00 --gres=gpu:1 python tensorflow_mnist.py
```

or with \texttt{sbatch} by submitting \texttt{tensorflow_mnist.sh}:

```bash
#!/bin/bash
#SBATCH --gres=gpu:1
#SBATCH --time=00:15:00
module load anaconda
python tensorflow_mnist.py
```

Do note that by default Keras downloads datasets to \texttt{$HOME/.keras/datasets}.

**Running simple Tensorflow/Keras model with NVIDIA’s containers**

Let’s run the MNIST example from Tensorflow’s tutorials:

```python
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

The full code for the example is in \texttt{tensorflow_mnist.py}. One can run this example with \texttt{srun}:
wget https://raw.githubusercontent.com/AaltoSciComp/scicomp-docs/master/triton/
  --examples/tensorflow/tensorflow_mnist.py
module load nvidia-tensorflow/20.02-tf1-py3
srun --time=00:15:00 --gres=gpu:1 singularity_wrapper exec python tensorflow_mnist.py

or with `sbatch` by submitting `tensorflow_singularity_mnist.sh`:

```bash
#!/bin/bash
#SBATCH --gres=gpu:1
#SBATCH --time=00:15:00
module load nvidia-tensorflow/20.02-tf1-py3
singularity_wrapper exec python tensorflow_mnist.py
```

Do note that by default Keras downloads datasets to `$HOME/.keras/datasets`.

**Common problems**

- **ImportError: libcuda.so.1: cannot open shared object file: No such file or directory.** Older versions of GPU tensorflow can only be imported on GPU nodes (even though you'd think that you can import it and just not use the GPUs). So you can only run this code in the GPU queue. Solution for this is to use the newer anaconda-modules.

- Random CUDA errors: don’t load any other CUDA modules, only `anaconda`. Anaconda includes the necessary libraries in compatible versions.

**Theano**

- **supportlevel**
- **pagelastupdated**
- **maintainer**

If you’re using the theano library, you need to tell theano to store compiled code on the local disk on the compute node. Create a file `~/.theanorc` with the contents

```
[global]
base_compiledir=/tmp/%(user)s/theano
```

Also make sure that in your batch job script you create this directory before you launch theano. E.g.

```bash
mkdir -p /tmp/${USER}/theano
```

The problem is that by default the `base_compiledir` is in your home directory (`~/.theano/`), and then if you first happen to run a job on a newer processor, a later job that happens to run on an older processor will crash with an “Illegal instruction” error.
VASP

VASP (Vienna Ab initio Simulation Package) is a computer program for atomic scale materials modelling, e.g. electronic structure calculations and quantum-mechanical molecular dynamics, from first principles.

VASP is licensed software, requiring the licensee to keep the vasp team updated with a list of user names. Thus, in order to use VASP arrange with the “vaspmaster” for your group to be put on the vasp licensed user list. Afterwards, contact your local triton admin who will take care of the IT gymnastics, and CC the vaspmaster so that he is aware of who gets added to the list.

For the PHYS department, the vaspmaster is Janne Blomqvist.

For each VASP version, there are 3 binaries compiled. All versions are MPI versions.

- vasp_std: The “standard” vasp, compiled with NGZhalf
- vasp_gam: Gamma point only. Faster if you use only a single k-point.
- vasp_ncl: For non-collinear spin calculations

VASP 5.4.4

The binaries are compiled with the Intel compiler suite and the MKL library, the used toolchain module is intel/2019a. Example batch script

```bash
#!/bin/bash -l
#SBATCH --nodes=1
#SBATCH --ntasks=8
#SBATCH --time=06:00:00
#SBATCH --mem-per-cpu=1500M
ml vasp/5.4.4
mpirun vasp_std
```

Note that contrary to our usual instructions where we strongly recommend to use `srun` to launch MPI applications, here we must use `mpirun` as the `srun` launcher does not work when using Intel MPI.

Potentials

Potentials are stored at `/share/apps/vasp/pot`.

Old VASP versions (obsolete, for reference only!)

These old versions are unlikely to work as they use old MPI and IB libraries that have stopped working due to upgrades over the years.
VASP 5.4.1

Currently the binaries are compiled with GFortran instead of Intel Fortran (the Intel Fortran binaries crashed, don’t know why yet). Example batch script

```bash
#!/bin/bash -l
#SBATCH --nodes=1
#SBATCH --ntasks=8
#SBATCH --time=06:00:00
#SBATCH --mem-per-cpu=1500M
module load vasp/5.4.1-gmvolf-triton-2016a
srun vasp_std
```

For each VASP version, there are two binaries compiled with slightly different options:

- `vasp.mpi.NGZhalf`
- `vasp.mpi`

Both are MPI versions. The first one is what you should normally use; it is compiled with the NGZhalf option which reduces charge density in the Z direction, leading to less memory usage and faster computation. The second version is needed for non-collinear spin calculations. The binaries can be found in the directory `/share/apps/vasp/$VERSION/`. For those of you who need to compile your own version of VASP, the makefiles used for these builds can be used as a starting point, and are found in the directory `/share/apps/vasp/makefiles`.

VASP 5.3.5

The binaries are optimized for the Xeon Ivy Bridge nodes, although they will also work fine on the older Xeon Westmere and Opteron nodes. Note that for the moment only the NGZhalf version has been built. If you need the non-NGZhalf version for non-collinear spin calculations please contact triton support. Example job script below:

```bash
#!/bin/bash -l
#SBATCH --nodes=1
#SBATCH --ntasks=12
#SBATCH --time=06:00:00
#SBATCH --mem-per-cpu=2500M
module load vasp/5.3.5
srun vasp.mpi.NGZhalf
```

The relative time to run the vasptest v2 testsuite on 12 cores (so a full node for Xeon Westmere and Opteron nodes, and 12/20 cores on a Xeon Ivy Bridge node) is for Xeon IB/Xeon Westmere/Opteron 1.0/2.0/2.8. So one sees that the Xeon Ivy Bridge nodes are quite a lot faster per core than the older nodes (with the caveat that the timings may vary depending on other jobs that may have been running on the Xeon IB node during the benchmark).
**VASP 5.3.3**

The binaries are optimized for the Xeon nodes, although they also work on the Opteron nodes. Some simple benchmarks suggest that the Opteron nodes are a factor of 1.5 slower than the Xeon nodes, although it is recommended to write the batch script such that Opteron nodes can also be used, as the Opteron queue is often shorter. An example script below:

```bash
#!/bin/bash -l
#SBATCH --nodes=1
#SBATCH --ntasks=12
#SBATCH --time=06:00:00
#SBATCH --mem-per-cpu=2500M

module load vasp/5.3.3
srun vasp.mpi.NGZhalf
```

**VASP 5.3.2 and older**

The binaries are optimized for the Intel Xeon architecture nodes, and are not expected to work on the Opteron nodes. An example job script is below (Note that it is different from the script for version 5.3.3 and newer above!):

```bash
#!/bin/bash -l
#SBATCH --nodes=1
#SBATCH --ntasks=12
#SBATCH --time=1-00:00:00
#SBATCH --mem-per-cpu=3500M

module load vasp/5.3.2
srun vasp.mpi.NGZhalf
```

**Potentials**

PAW potentials for VASP can be found in the directory /share/apps/vasp/pot. The recommended potentials are the ones in the Apr2012.52 subdirectory. For reference, an older set of potentials dating back to 2003 can be found in the “2003” subdirectory.
Validation

The vasp.mpi.NGZhalf builds have been verified to pass all the tests in the vasptest suite.

Other

Old makefiles

Here is a number of Makefiles copy-pasted from old Rocks installation. Can be useful in general, though may require adaptation to new installation. Please, send us a fully working copy if you have one.

See old wiki: https://wiki.aalto.fi/display/Triton/Applications

Rename vasp.x.y.makefile => vasp.x.y/makefile

VisIT

This uses Singularity containers, so you should refer to that page first for general information.

Visit has been compiled using the build_visit-script from the VisIT page on an Ubuntu image. It has minimal amount of other software installed.

Parallelization is done against Triton’s OpenMPI, so using this container with other OpenMPI modules is discouraged.

Within the container VisIT is installed under /opt/visit/. PATH is automatically appended with their respective paths so all program calls are available automatically.

Usage

This example shows how you can launch visit on the login node for small visualizations or launch it in multiprocess state on a reserved node. Firstly, let’s load the module:

```
module use /share/apps2/singularity/modules
module load Visit
```

Now you can run VisIT with:

```
singularity_wrapper exec visit
```

If you want to run VisIT with multiple CPUs, you should reserve a node with sinteractive:

```
sinteractive --time=00:30:00 --ntasks=2 --nodes=1-1
singularity_wrapper exec visit -np 2
```

Do note the flag --nodes=1-1 that ensures that all of VisITs processes end up on the same node. Currently VisIT encounters problems when going across the node lines.
6.1.6 Reference and Examples

Triton quick reference

In this page, you have all important reference information

Modules

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module load NAME</td>
<td>load module</td>
</tr>
<tr>
<td>module avail</td>
<td>list all modules</td>
</tr>
<tr>
<td>module spider NAME</td>
<td>search modules</td>
</tr>
<tr>
<td>module list</td>
<td>list currently loaded modules</td>
</tr>
<tr>
<td>module show NAME</td>
<td>details on a module</td>
</tr>
<tr>
<td>module help NAME</td>
<td>details on a module</td>
</tr>
<tr>
<td>module unload NAME</td>
<td>unload a module</td>
</tr>
<tr>
<td>module save ALIAS</td>
<td>save module collection to this alias (saved in ~/.lmod.d/)</td>
</tr>
<tr>
<td>module restore ALIAS</td>
<td>load saved module collection (faster than loading individually)</td>
</tr>
<tr>
<td>module purge</td>
<td>unload all loaded modules (faster than unloading individually)</td>
</tr>
</tbody>
</table>

Common software

- **Python**: module load anaconda for the Anaconda distribution of Python 3, including a lot of useful packages. [More info.](#)
- **R**: module load r for a basic R package. [More info.](#)
- **Matlab**: module load matlab for the latest Matlab version. [More info.](#)
- **Julia**: module load julia for the latest Julia version. [More info.](#)
Storage

<table>
<thead>
<tr>
<th>Name</th>
<th>Path</th>
<th>Quota</th>
<th>Backup</th>
<th>Locality</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
<td>$HOME or /home/$username/</td>
<td>hard quota 10GB</td>
<td>Nightly</td>
<td>all nodes</td>
<td>Small user specific files, no calculation data.</td>
</tr>
<tr>
<td>Work</td>
<td>$WRKDIR or /scratch/work/$username/</td>
<td>200GB and 1 million files</td>
<td>x</td>
<td>all nodes</td>
<td>Personal working space for every user. Calculation data etc. Quota can be increased on request.</td>
</tr>
<tr>
<td>Scratch</td>
<td>/scratch/$dept/$project/</td>
<td>on request</td>
<td>x</td>
<td>all nodes</td>
<td>Department/group specific project directories.</td>
</tr>
<tr>
<td>Local temp</td>
<td>/tmp/</td>
<td>limited by disk size</td>
<td>x</td>
<td>single-node</td>
<td>Primary (and usually fastest) place for single-node calculation data. Removed once user’s jobs are finished on the node.</td>
</tr>
<tr>
<td>Local persistent</td>
<td>/1/</td>
<td>varies</td>
<td>x</td>
<td>dedicated group servers only</td>
<td>Local disk persistent storage. On servers purchased for a specific group. Not backed up.</td>
</tr>
<tr>
<td>ramfs (login nodes only)</td>
<td>$XDG_RUNTIME_DIR</td>
<td>limited by memory</td>
<td>x</td>
<td>single-node</td>
<td>Ramfs on the login node only, in-memory filesystem</td>
</tr>
</tbody>
</table>

Partitions

<table>
<thead>
<tr>
<th>Partition</th>
<th>Max job size</th>
<th>Mem/core (GB)</th>
<th>Tot mem (GB)</th>
<th>Cores/node limits</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;default&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If you leave off all possible partitions will be used (based on time/mem)</td>
</tr>
<tr>
<td>debug</td>
<td>2 nodes</td>
<td>2.66 - 12</td>
<td>32-256</td>
<td>12,20,24</td>
<td>15 min testing and debugging short interactive. work. 1 node of each arch.</td>
</tr>
<tr>
<td>batch</td>
<td>16 nodes</td>
<td>2.66 - 12</td>
<td>32-256</td>
<td>12, 20,24</td>
<td>5d primary partition, all serial &amp; parallel jobs</td>
</tr>
<tr>
<td>short</td>
<td>8 nodes</td>
<td>4 - 12</td>
<td>48-256</td>
<td>12, 20,24</td>
<td>4h short serial &amp; parallel jobs, +96 dedicated CPU cores</td>
</tr>
<tr>
<td>huge-mem</td>
<td>1 node</td>
<td>43</td>
<td>1024</td>
<td>24</td>
<td>3d huge memory jobs, 1 node only</td>
</tr>
<tr>
<td>gpu</td>
<td>1 node, 2-8GPUs</td>
<td>2 - 10</td>
<td>24-128</td>
<td>12</td>
<td>5d GPU computing</td>
</tr>
<tr>
<td>gpushort</td>
<td>4 nodes, 2-8GPUs</td>
<td>2 - 10</td>
<td>24-128</td>
<td>12</td>
<td>4h GPU computing</td>
</tr>
<tr>
<td>interactive</td>
<td>2 nodes</td>
<td>5</td>
<td>128</td>
<td>24</td>
<td>1d for sinteractive command, longer interactive work</td>
</tr>
</tbody>
</table>

Use slurm partitions to see more details.
## Job submission

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbatch</td>
<td>submit a job to queue (see standard options below)</td>
</tr>
<tr>
<td>srun</td>
<td>Within a running job script/environment: Run code using the allocated resources (see options below)</td>
</tr>
<tr>
<td>srun</td>
<td>On frontend: submit to queue, wait until done, show output. (see options below)</td>
</tr>
<tr>
<td>sinteractive</td>
<td>Submit job, wait, provide shell on node for interactive playing (X forwarding works, default partition interactive). Exit shell when done. (see options below)</td>
</tr>
<tr>
<td>srun</td>
<td>(advanced) Another way to run interactive jobs, no X forwarding but simpler. Exit shell when done.</td>
</tr>
<tr>
<td>srun --pty bash</td>
<td></td>
</tr>
<tr>
<td>scancel &lt;jobid&gt;</td>
<td>Cancel a job in queue</td>
</tr>
<tr>
<td>salloc</td>
<td>(advanced) Allocate resources from frontend node. Use srun to run using those resources, exit to close shell when done. Read the description! (see options below)</td>
</tr>
<tr>
<td>scontrol</td>
<td>View/modify job and slurm configuration</td>
</tr>
</tbody>
</table>

6.1. Triton cluster
<table>
<thead>
<tr>
<th>Command/Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| sbatch | time limit
| --time=hh:mm:ss | time limit, days-hours
| -t, --time=dd-hh | job partition. Usually leave off and things are auto-detected.
| -p, --partition=partition | request n MB of memory per core
| --mem-per-cpu=n | request n MB memory per node
| -c, --cpus-per-task=n | Allocate *n* CPU's for each task. For multithreaded jobs. (compare `-ntasks`: `-c` means the number of cores for each process started.)
| -N, --nodes=n-n | allocate minimum of n, maximum of m nodes.
| -n, --ntasks=n | allocate resources for and start n tasks (one task=one process started, it is up to you to make them communicate. However the main script runs only on first node, the sub-processes run with `srun` are run this many times.)
| -J, --job-name=name | short job name
| -o output | print output into file `output`
| -e error | print errors into file `error`
| --exclusive | allocate exclusive access to nodes. For large parallel jobs.
| --constraint=feature | request a feature (see `slurm features` for the current list of configured features, or `Arch under the hardware list`). Multiple with `--constraint="hsw|skl"`.
| --array=0-5,7,10-15 | Run job multiple times, use variable `$SLURM_ARRAY_TASK_ID` to adjust parameters.
| --gres=gpu | request a GPU, or `--gres=gpu:n` for multiple
| --gres=spindle:disk | request nodes that have disks, `spindle:n`, for a certain number of RAID0 disks
| --mail-type=type | notify of events: BEGIN, END, FAIL, ALL, REQUEUE (not on triton) or ALL. MUST BE used with `--mail-user=only`
| --mail-user=your@email | who to send the email
| srun | Print allocated nodes (from within script)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| `slurm q` | Status of your queued jobs (long/short)
| `slurm partitions` | Overview of partitions (A/I/O/T=active,idle,other,total)
| `slurm cpus <partition>` | list free CPUs in a partition
| `slurm history [1day,2hour,...]` | Show status of recent jobs
| `seff <jobid>` | Show percent of mem/CPU used in job
| `slurm j <jobid>` | Job details (only while running)
| `slurm s;slurm ss <partition>` | Show status of all jobs
| `sacct` | Full history information (advanced, needs args)

Full slurm command help:

```bash
$ slurm
```

(continues on next page)
Show or watch job queue:

- **slurm [watch] queue** — show own jobs
- **slurm [watch] q** — show user's jobs
- **slurm [watch] quick** — show quick overview of own jobs
- **slurm [watch] shorter** — sort and compact entire queue by job size
- **slurm [watch] short** — sort and compact entire queue by priority
- **slurm [watch] full** — show everything
- **slurm [w] [q|qq|ss|s|f]** — shorthands for above!
- **slurm qos** — show job service classes
- **slurm top [queue|all]** — show summary of active users

Show detailed information about jobs:

- **slurm prio [all|short]** — show priority components
- **slurm j|job** — show everything else
- **slurm steps** — show memory usage of running srun job steps

Show usage and fair-share values from accounting database:

- **slurm h|history** — show jobs finished since, e.g. "1day" (default)
- **slurm shares**

Show nodes and resources in the cluster:

- **slurm p|partitions** — all partitions
- **slurm n|nodes** — all cluster nodes
- **slurm c|cpus** — total cpu cores in use
- **slurm cpus** — cores available to partition, allocated and free
- **slurm cpus jobs** — cores/memory reserved by running jobs
- **slurm cpus queue** — cores/memory required by pending jobs
- **slurm features** — list features and GRES

Examples:

- **slurm q**
- **slurm watch shorter**
- **slurm cpus batch**
- **slurm history 3hours**

**Other advanced** commands (many require lots of parameters to be useful):

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>squeue</td>
<td>Full info on queues</td>
</tr>
<tr>
<td>slinfo</td>
<td>Advanced info on partitions</td>
</tr>
<tr>
<td>slurm nodes</td>
<td>List all nodes</td>
</tr>
</tbody>
</table>
The table below lists the toolchains available for use with Aalto scientific computing.
## Hardware

<table>
<thead>
<tr>
<th>Node name</th>
<th>Number of nodes</th>
<th>Node type</th>
<th>Year</th>
<th>Arch (constraint)</th>
<th>CPU type</th>
<th>Memory Configuration</th>
<th>Infini-band</th>
<th>GPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivb[1-24]</td>
<td>24</td>
<td>ProLiant SL230s G8</td>
<td></td>
<td>ivb avx</td>
<td>2x10 core Xeon E5 2680 v2 2.80GHz</td>
<td>256GB DDR3-1667</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>ivb[25-48]</td>
<td>24</td>
<td>ProLiant SL230s G8</td>
<td></td>
<td>ivb avx</td>
<td>2x10 core Xeon E5 2680 v2 2.80GHz</td>
<td>64GB DDR3-1667</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>pe[1-48, 65-81]</td>
<td>65</td>
<td>Dell PowerEdge C4130</td>
<td>2016</td>
<td>hsw avx avx2</td>
<td>2x12 core Xeon E5 2680 v3 2.50GHz</td>
<td>128GB DDR4-2133</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>pe[49-64,82]</td>
<td>17</td>
<td>Dell PowerEdge C4130</td>
<td>2016</td>
<td>hsw avx avx2</td>
<td>2x12 core Xeon E5 2680 v3 2.50GHz</td>
<td>256GB DDR4-2133</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>pe[83-911]</td>
<td>8</td>
<td>Dell PowerEdge C4130</td>
<td>2017</td>
<td>bdw avx avx2</td>
<td>2x14 core Xeon E5 2680 v4 2.40GHz</td>
<td>128GB DDR4-2400</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>c[579-628,639-698]</td>
<td>110</td>
<td>ProLiant XL230a Gen9</td>
<td>2017</td>
<td>hsw avx avx2</td>
<td>2x12 core Xeon E5 2690 v3 2.60GHz</td>
<td>128GB DDR4-2666</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>c[629-638]</td>
<td>10</td>
<td>ProLiant XL230a Gen9</td>
<td>2017</td>
<td>hsw avx avx2</td>
<td>2x12 core Xeon E5 2690 v3 2.60GHz</td>
<td>256GB DDR4-2400</td>
<td>FDR</td>
<td></td>
</tr>
<tr>
<td>skl[1-48]</td>
<td>48</td>
<td>Dell PowerEdge C6420</td>
<td>2019</td>
<td>skl avx avx2 avx512</td>
<td>2x20 core Xeon Gold 6148 2.40GHz</td>
<td>192GB DDR4-2667</td>
<td>EDR</td>
<td></td>
</tr>
<tr>
<td>csl[1-48]</td>
<td>48</td>
<td>Dell PowerEdge C6420</td>
<td>2020</td>
<td>csl avx2 avx512</td>
<td>2x20 core Xeon Gold 6248 2.50GHz</td>
<td>192GB DDR4-2667</td>
<td>EDR</td>
<td></td>
</tr>
<tr>
<td>fn3</td>
<td>1</td>
<td>Dell PowerEdge R940</td>
<td>2020</td>
<td>avx avx2 avx512</td>
<td>4x20 core Xeon Gold 6148 2.40GHz</td>
<td>2TB DDR4-2666</td>
<td>EDR</td>
<td></td>
</tr>
<tr>
<td>gpu[1-10]</td>
<td>10</td>
<td>Dell PowerEdge C4140</td>
<td>2020</td>
<td>skl avx2 avx512 volta</td>
<td>2x8 core Intel Xeon Gold 6134 @ 3.2GHz</td>
<td>384GB DDR4-2667</td>
<td>EDR</td>
<td></td>
</tr>
<tr>
<td>gpu[20-22]</td>
<td>3</td>
<td>Dell PowerEdge C4130</td>
<td>2016</td>
<td>hsw avx2 Kepler</td>
<td>2x6 core Xeon E5 2620 v3 2.50GHz</td>
<td>128GB DDR4-2133</td>
<td>EDR</td>
<td></td>
</tr>
<tr>
<td>gpu[23-27]</td>
<td>5</td>
<td>Dell PowerEdge C4130</td>
<td>2017</td>
<td>hsw avx2 Pascal</td>
<td>2x12 core Xeon E5 2680 v3 @ 2.5GHz</td>
<td>256GB DDR4-2400</td>
<td>EDR</td>
<td></td>
</tr>
<tr>
<td>dgx[01-02]</td>
<td>2</td>
<td>Nvidia DGX-1</td>
<td>2018</td>
<td>bdw avx2 volta</td>
<td>2x20 core Xeon E5 2698 v4 @ 2.2GHz</td>
<td>512GB DDR4-2133</td>
<td>EDR</td>
<td></td>
</tr>
<tr>
<td>gpu[28-37]</td>
<td>10</td>
<td>Dell PowerEdge C4140</td>
<td>2019</td>
<td>skl avx2 avx512 volta</td>
<td>2x8 core Intel Xeon Gold 6134 @ 3.2GHz</td>
<td>384GB DDR4-2667</td>
<td>EDR</td>
<td></td>
</tr>
</tbody>
</table>

### 6.1. Triton cluster
Aalto scientific computing guide

<table>
<thead>
<tr>
<th>Node type</th>
<th>CPU count</th>
</tr>
</thead>
<tbody>
<tr>
<td>48GB Xeon Westmere (2012)</td>
<td>1404</td>
</tr>
<tr>
<td>24GB Xeon Westmere + 2x GPU (2012)</td>
<td>120</td>
</tr>
<tr>
<td>96GB Xeon Westmere (2012)</td>
<td>288</td>
</tr>
<tr>
<td>1TB Xeon Westmere (2012)</td>
<td>48</td>
</tr>
<tr>
<td>256GB Xeon Ivy Bridge (2014)</td>
<td>480</td>
</tr>
<tr>
<td>64GB Xeon Ivy Bridge (2014)</td>
<td>480</td>
</tr>
<tr>
<td>128GB Xeon Haswell (2016)</td>
<td>1224</td>
</tr>
<tr>
<td>256GB Xeon Haswell (2016)</td>
<td>360</td>
</tr>
<tr>
<td>128GB Xeon Haswell + 4x GPU (2016)</td>
<td>36</td>
</tr>
</tbody>
</table>

**GPUs**

<table>
<thead>
<tr>
<th>Card</th>
<th>total amount</th>
<th>nodes</th>
<th>architecture</th>
<th>compute threads per GPU</th>
<th>memory per card</th>
<th>CUDA compute capability</th>
<th>Slurm feature name</th>
<th>Slurm gres name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tesla K80*</td>
<td>12</td>
<td>gpu[20-22]</td>
<td>Kepler</td>
<td>2x2496</td>
<td>2x12GB</td>
<td>3.7</td>
<td>kepler</td>
<td>teslak80</td>
</tr>
<tr>
<td>Tesla P100</td>
<td>20</td>
<td>gpu[23-27]</td>
<td>Pascal</td>
<td>3854</td>
<td>16GB</td>
<td>6.0</td>
<td>pascal</td>
<td>teaplap100</td>
</tr>
<tr>
<td>Tesla V100</td>
<td>40</td>
<td>gpu[1-10]</td>
<td>Volta</td>
<td>5120</td>
<td>32GB</td>
<td>7.0</td>
<td>volta</td>
<td>v100</td>
</tr>
<tr>
<td>Tesla V100</td>
<td>40</td>
<td>gpu[28-37]</td>
<td>Volta</td>
<td>5120</td>
<td>32GB</td>
<td>7.0</td>
<td>volta</td>
<td>v100</td>
</tr>
<tr>
<td>Tesla V100</td>
<td>16</td>
<td>dgx[01-02]</td>
<td>Volta</td>
<td>5120</td>
<td>16GB</td>
<td>7.0</td>
<td>volta</td>
<td>v100</td>
</tr>
</tbody>
</table>

**Examples**

**Master-Worker Example**

Following example shows how to manage host list using the python-hostlist package and run different tasks for master task and worker task.

This kind of structure might be needed if one wants to create a e.g. Spark cluster or use some other program that uses master-worker-paradigm, but does not use MPI.

It is important to make sure that in case of job cancellation all programs started by the scripts will be killed gracefully. In case of Spark or other programs that initialize a cluster using SSH and then forking a process, these forked processes must be killed after job allocation has ended.

hostlist-test.slrm:

```bash
#!/bin/bash
#SBATCH --time=00:10:00
#SBATCH --nodes=3
#SBATCH --ntasks=5
#SBATCH --o hostlist-test.out

# An example of a clean_up-routine if the master has to take e.g. ssh connection to...
```

(continues on next page)
function clean_up {
    echo "Got SIGTERM, will clean up my workers and exit."
    exit
}
trap clean_up SIGHUP SIGINT SIGTERM

# Actual script that defines what each worker will do
srun bash run.sh

run.sh:

#!/bin/bash

# Get a list of hosts using python-hostlist
nodes=`hostlist --expand $SLURM_NODELIST|xargs`

# Determine current worker name
me=$(hostname)

# Determine master process (first node, id 0)
master=$(echo $nodes | cut -f 1 -d ' ')

# SLURM_LOCALID contains task id for the local node
localid=$SLURM_LOCALID

if [[ "$me" == "$master" && "$localid" -eq 0 ]]
then
    # Run these if the process is the master task
    echo "I'm the master with number "$localid" in node "${me}". My subordinates are "
    echo $nodes
else
    # Run these if the process is a worker
    echo "I'm a worker number "$localid" in node "${me}"
fi

Example output:

I'm a worker number 1 in node opt469
I'm a worker number 2 in node opt469
I'm the master with number 0 in node opt469. My subordinates are opt469 opt470 opt471
I'm a worker number 0 in node opt471
I'm a worker number 0 in node opt470

Python OpenMP example

parallel_Python.slrm:

#!/bin/bash
#SBATCH --time=00:10:00
#SBATCH --cpus-per-task=4
#SBATCH --mem=2G
#SBATCH -o parallel_Python.out
module load anaconda
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
srun -c $SLURM_CPUS_PER_TASK python parallel_Python.py

parallel\Python.py:

```python
import numpy as np
a = np.random.random([2000,2000])
a = a + a.T
b = np.linalg.pinv(a)
print(np.amax(np.dot(a,b)))
```

Serial R example

r_serial.slrm:

```bash
#!/bin/bash -l
#SBATCH --time=00:05:00
#SBATCH --ntasks=1
#SBATCH --mem=100M
#SBATCH --output=r_serial.out

module load r
n=3
m=2
srun Rscript --vanilla r_serial.R $n $m
```

r_serial.R:

```r
args = commandArgs(trailingOnly=TRUE)
n<-as.numeric(args[1])
m<-as.numeric(args[2])
print(n)
print(m)
A<-t(matrix(0:5,ncol=n,nrow=m))
print(A)
B<-t(matrix(2:7,ncol=n,nrow=m))
print(B)
C<-matrix(0.5,ncol=n,nrow=n)
print(C)
C<-A %*% t(B) + 2*C
print(C)
```
Parallel R example

r_parallel.slrm:

```bash
#!/bin/bash
#SBATCH --time=00:20:00
#SBATCH --cpus-per-task=4
#SBATCH --mem=2G
#SBATCH --output=r_parallel.out

# Set the number of OpenMP-threads to 1,
# as we're using parallel for parallelization
export OMP_NUM_THREADS=1

# Load the version of R you want to use
module load r

# Run your R script
srun Rscript r_parallel.R
```

```r
library(pracma)
library(parallel)
invertRandom <- function(index) {
  A<-matrix(runif(2000*2000),ncol=2000,nrow=2000);
  A<-A + t(A);
  B<-pinv(A);
  return(max(B %*% A));
}
ptm<-proc.time()
mclapply(1:16,invertRandom, mc.cores=Sys.getenv('SLURM_CPUS_PER_TASK'))
proc.time()-ptm
```

When constrained to opt-architecture, run times for different core numbers were

<table>
<thead>
<tr>
<th>ncores</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>runtime</td>
<td>380.757</td>
<td>182.185</td>
<td>125.526</td>
<td>84.230</td>
</tr>
</tbody>
</table>

R OpenMP Example

r_openmp.slrm:

```bash
#!/bin/bash
#SBATCH --time=00:15:00
#SBATCH --cpus-per-task=4
#SBATCH --mem=2G
#SBATCH --output=r_openmp.out

module load r
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

time srun Rscript --default-packages=methods,utils,stats R-benchmark-25.R
```

The benchmark script is available here (more information about it is available here page).
Python

IPython parallel

A example batch script that uses IPython parallel (ipyparallel) within slurm. See also the interactive hints on the Python page.

ipyparallel uses global state in your home directory, so you can only run one of these at a time! You can add the --profile= option to name different scripts (you could use $SLURM_JOB_ID). But then you will get a growing number of unneeded profile directories at ~/.ipython/profile_*, so this isn’t recommended. Basically, ipyparallel is more designed for one-at-a-time interactive use rather than batch scripting (unless you do more work…).

ipyparallel.slrm is an example slurm script that sets up ipyparallel. It assumes that most work is done in the engines. It has inline Python, replace this with python your_script_name.py

```bash
#!/bin/bash
#SBATCH --nodes=4

module load anaconda
set -x

ipcontroller --ip="*" &
sleep 5
# Run the engines in slurm job steps (makes four of them, since we use
# the --nodes=4 slurm option)... 
srun ipengine --location=$(hostname -f) &

sleep 5
# Put the actual Python isn't in a job step. This is assuming that
# most work happens in engines
python3 <<EOF
import os
import ipyparallel
client = ipyparallel.Client()
result = client[:].apply_async(os getpid)
pid_map = result.get_dict()
print(pid_map)
EOF
```

Python MPI4py

A simple script mpi4py.py that utilizes mpi4py.

```python
#!/usr/bin/env python

""
Parallel Hello World
""

from mpi4py import MPI
import sys
size = MPI.COMM_WORLD.Get_size()
rank = MPI.COMM_WORLD.Get_rank()
name = MPI.Get_processor_name()
sys.stdout.write("Hello, World! I am process \$d of \$d on \$s.\n"
% (rank, size, name))
```
Running mpi4py.py using only srun:

```bash
module load Python/2.7.11-goolf-triton-2016b
srun --time=00:10:00 --ntasks=4 -p debug python helloworld.py
```

Example sbatch script mpi4py.slrm when running mpi4py.py through sbatch:

```bash
#!/bin/bash
#SBATCH --time=00:10:00
#SBATCH --ntasks=4

module load Python/2.7.11-goolf-triton-2016b
mpiexec -n $SLURM_NTASKS python mpi4py.py
```

### Running Python with OpenMP parallelization

Various Python packages such as Numpy, Scipy and pandas can utilize OpenMP to run on multiple CPUs. As an example, let’s run the python script `python_openmp.py` that calculates multiplicative inverse of five symmetric matrices of size 2000x2000.

```python
nrounds = 5

t_start = time()

for i in range(nrounds):
    a = np.random.random([2000,2000])
    a = a + a.T
    b = np.linalg.pinv(a)

    t_delta = time() - t_start

print('Seconds taken to invert %d symmetric 2000x2000 matrices: %f' % (nrounds, t_delta))
```

The full code for the example is in HPC examples-repository. One can run this example with `srun`:

```bash
wget https://raw.githubusercontent.com/AaltoSciComp/hpc-examples/master/python/python_openmp/python_openmp.py
module load anaconda
export OMP_PROC_BIND=true
srun --cpus-per-task=2 --mem=2G --time=00:15:00 python python_openmp.py
```

or with `sbatch` by submitting `python_openmp.slrm`:

```bash
#!/bin/bash -l
#SBATCH --time=00:10:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=2
#SBATCH --mem-per-cpu=1G
#SBATCH -o python_openmp.out

module load anaconda/2020-03-tf2
export OMP_PROC_BIND=true

echo 'Running on: '$HOSTNAME
```

(continues on next page)
Important: Python has a global interpreter lock (GIL), which forces some operations to be executed on only one thread and when these operations are occurring, other threads will be idle. These kinds of operations include reading files and doing print statements. Thus one should be extra careful with multithreaded code as it is easy to create seemingly parallel code that does not actually utilize multiple CPUs.

There are ways to minimize effects of GIL on your Python code and if you’re creating your own multithreaded code, we recommend that you take this into account.

Cheatsheets: Triton
Skills to do science are different than skills to write good research code. The Aalto Research Software Engineering group provides support and mentoring to those using computing and data.

### 7.1 Research Software Engineers

The Aalto Research Software Engineers (RSEs) provide specialist support regarding software, computing, and data. As research becomes more digital and computer-dependent, the prerequisite knowledge grows larger and larger, and we exist to help you fill that gap.

For example, we can do:

- Custom software development or modification
- Scaling up to larger computing systems, code optimization, workflow automation
- Preparing a code for release, or cleaning up old code that needs to be re-used
- Transforming a code from an individual project to a collaborative project (or group training on this topic)
- Data services (optimizing I/O, data versioning, databases, data sharing, and opening)

Instead of hiring your own intern, postdoc, etc. to struggle with certain issues, we can help instead. We consist of experienced researchers who have relied heavily on computing technology (programming, computing, data) for our academic work, and thus can seamlessly collaborate on research projects. We can also do consultation and training. You will have more impact since your work is more reusable, open, and higher quality.

Departments can join this program and make this service to its community as a basic service. Current member departments are CS, NBE, PHYS, and SCI overall.

**Contact**

For a quick chat, come to our *daily garage*, every day online at 13:00 (Tuesdays are RSE focus days). Or contact us by email at rse-group at aalto.fi, or fill out our request form. See *requesting RSE* for more.
7.1.1 About our services

For researchers and research groups

You program or analyze data in your daily work, and you know something is missing: your code and data is less organized, less efficient, less managed than others, and it’s affecting the quality of your work. Or maybe you don’t know how to start your project, or publish it. You’re too busy with the science to have time to focus on the computing.

To request a service, contact Aalto Scientific Computing or answer our survey and request to be contacted this form.

Case study: preparation for publication

A group is about to publish a paper about a method, but their code is a bit messy. Without easy-to-use, (relatively) high-quality code, they know their impact will be minimal. They invest in a few days of RSE work in order to help adopt best practices and release their method as open source.

Case study: external grant

A PI has gotten a large external grant, and as part of that they need some software development expertise. The time frame is four months, but they can’t hire a top-quality person on an academic salary for that short time. They contact the Aalto RSE group (either before the grant, or while it is running) and use our speciality for four days per week.

Case study: improve workflow

A group of researchers all works on similar things, but independently since their backgrounds have been in science, not software development. They invite the RSE for a quick consultation to help them get set up with version control and show a more modular way to structure their code, so they can start some real collaborations, not just talking. This is the first step to more impact (and open science) from their work.

Case study: sustainability of finished projects

A project has ended and the main person who managed the code/analysis pipeline has left to continue their career somewhere else. You wish to replicate and extend the previous work, but your only starting point is a folder with hundreds of files and no clear instructions/documentation. Aalto RSEs can help you re-using and recycling previous code, document it, and extend it to make it more sustainable to be reused in future projects.

What we do

Our RSEs may provide both mentorship and programming-as-a-service for your projects. Are you tired of research being held back by slow programming? We can help.

You can request our help for a few hours, to consult and co-work on a project. Our goal will be to primarily teaching and mentoring, to help you help yourselves in the long run. We’ll point you in the right direction and where to look next.

You can also request longer-term programming as a service. This can be producing or modifying some software for you, or whatever you may need. This is expected to be paid from your grants. (Need someone for a few months for your grant? We can do that.)
Short-term examples

Format could be personal work, lecture, or group seminar followed by a hands-on session, for example.

- Setting up a project in version control with all the features. This also includes version control of data.
- Preparing code or data for release and publication
- FAIR data (findable, accessible, interoperable, reusable) - consultation and help.
- Creating or automating a workflow, especially those processing data or running simulations
- Optimizing some code - both for speed and/or adaptability
- Efficiency storing data for intensive analysis. Data replication and management.
- Making existing software more modular and reusable
- Help properly using, for example, machine learning library pipelines, instead of hacking things together yourself
- Setting up automatic software testing
- Transforming projects from individual to collaborative - either within a group, or open source.
- Generalized “code clean-up” which takes a project from development to stabilized

More involved examples

These would combine co-working, mentoring, and independent work. We go to you and work with you.

- Developing or maintaining specific software, services, demos, or implementations.
- Software development as a service
- Software support that lasts beyond the time frame of a single student’s attention
- Adding features to existing software
- Contributing to some other open source software you need for your research

Paid project service

In the dedicated service, your research groups pays and we will do whatever you want (in particular the more involved examples above). Still, our model is as much co-working as consulting: we want to improve your own skills so that you can still be productive afterwards.

The research group must pay for this service, but the rate is essentially at-cost and with minimal bureaucratic overhead.

Free basic service

In order to help everyone and avoid microtransactions, departments/schools/etc can sponsor a basic service, which provides a few hours or days of close support to improve how you work (especially for the “basic examples” above.

One of our trained RSEs will work with you for a short period to begin or improve your project. The goal is not to do it for you, but to show you by example so that you can do it yourself later.
How to contact us and request help

To request a service, see the request area. Requests are prioritized according to:

• Strategic benefit
• Long-term impact to research (for example, improved skills)
• Priority for units which provide funding
• Diversity and balance

For units such as departments

Our pilot is funded by departments and schools, and members of these units can receive the basic services free of charge (in accordance to the shares of funding). In addition to the basic service, researchers can request dedicated advanced services (see list above). Advanced services are funded directly by the research group and their funding replaces part of their unit’s share.

Case study: Systematic improvements

Your department has a lot of people doing little bits of programming everywhere, but everyone is doing things alone. What if they could work together better? By joining the RSE program as a unit, your staff can get up to X hours of free help to understand tools to make their programming/data work better. After a few years, you notice a dramatic cultural shift: there is more collaboration and higher-quality work. Perhaps you already see a change in your KPIs.

Benefits

Benefits to schools/departments:

• Increase the quality of efficiency of your research. Your researchers focus on their science while improving their coding skills passively.
• Provide hands-on technical research services to your community at a higher level than basic IT.
• Help with data management and FAIR data - be more competitive for funding, help get value out of your unit’s data.

Benefits to groups:

• Receive staff/on-call software development expertise within your group, without having to fund or hire a full-time.
• Your RSEs get joint training and share competence within our RSE group.
How to join

The RSE program is a part of Aalto Science-IT (Aalto Scientific Computing), so is integrated to our computing and data management infrastructure and training programs. You don’t just get a service, but a connection to a community of experts.

If you would like to join, check out the implementation plan and/or contact us at rse-group at aalto.fi

7.1.2 How to get started

Contact us as mentioned above, or read here for more details.

Requesting RSE support

You can contact us regardless of how small your issue is - or even if you would like to know if we could help your project. At least, we can point you in the right direction.

Quick Consultations

We recommend you come to our daily garage sessions for a short chat. There are no reservations, and this is the online equivalent of dropping by our office to say hi.

Contact

Our email is rse-group@aalto.fi (the Triton email address scicomp@aalto.fi also gets to us). Let us know what is going on, and we will find a good time to meet.

You can also use the structured request form (“Research Software Engineer request”). This guides you through some of our initial questions, but goes to the same place as email and everything is read by a human anyway.

Next steps

See How we work for more info.

How we work

This page is mostly focused on how long-term scheduled projects, which are funded by the research groups themselves, work. Long-term projects are scheduled by fraction of full-time-equivalent (FTE) over weeks or months.

For short-term code review tasks, come to any of our garage sessions and we will immediately take a look.
Types of service

- **Long-term** service deals with jobs that last months, and are scheduled in terms of FTE percentage over months. This is often directly as salary from some grant, as a researcher would be.

- **Medium-term** service deal with jobs scheduled in days. For simplicity, these are often fee-for-service paid internally from basic funding. Depending on your unit, they may also be free (paid by unit basic funding). *This is not worked out yet - check back for details*

- **Short-term** could be a code review at one of our *garages* or a few hours of work. This is generally free (paid by unit basic funding).

Beginning

Check if it is a type of task that we can do: best to ask us (TODO: improve this description later)

To actually make a request for support, see *Requesting RSE support*.

Initial meeting

First, you can expect an quick initial meeting between the researchers and RSEs. Depending on the size and complexity of the project, there may be several to find the right RSE and ensure that we can do a good job helping you.

- What scientific background knowledge is needed? How long does it take to get started?
- What type of contribution will the RSE make (see next section)? For purposes of scientific integrity, consider if this reaches the point of scientific authorship (see bottom).
- Researchers: provide access to code, documentation, and relevant scientific background in advance, so that they can be browsed. The more we know in advance, the better we can estimate the time required and how to best help you.
- How do you manage your data? To map things out, consider this one-page data management plan table.
- Final outputs, location, publication.
- Time frame and schedule flexibility.

What we can accomplish

It is very important to consider what the practical outcome of the project will be, because different researchers have very different needs. Together, we will think about these questions:

- What’s the object of focus
  - Software
  - Data
  - Workflows
- What is accomplished?
  - Create a brand-new product based on scientific specification. Is this done in an agile way (continuous feedback) or is the exact result known?
  - Improve some existing workflow or software, possibly drastically.
  - Improve some other project, primarily maintained by someone else.
– Prepare a project for publication, release, or being used by more people.

• Future plan
  – Primarily teach via example, so that the researcher can fully continue developing the project themselves.
  – Provide a finished product, which won’t need updates later
  – Provide a product that will be continually maintained by specialists (RSEs or similar - us?).

**Scheduling and planning**

RSEs will be assigned based on discussion between the researchers, RSEs, and Aalto Scientific Computing (the RSE group). Your agreement is with the RSE group, so your RSEs may change (even though we’ll try to avoid this).

We can never promise specific results in a specific time: we always agree based on a certain amount of time. As you can expect, projects can sometimes take far longer than expected, so we try to budget plenty of buffer into projects to accomplish the mandatory tasks.

Our exact scheduling system is not yet decided: if you start now, you help design the system.

**Costs and time tracking**

Right now, most service is free for all users within the School of Science.

TODO: future updates

**Funding practicalities**

Right now, most service is free to all users in the School of Science. In the future, longer-term projects can be funded by internal invoicing or directly hiring our RSE onto your grant, the same as salary of any researcher would work.

TODO: This will be developed as we begin the program. See above for a description.

**Getting started**

**Version control**

One can hardly do development work without using a good version control system. Our first step will be help you start using a version control system, if you are not yet using one, or if you are ensure you are using it optimally. If you don’t have a preference, we’ll recommend git and GitHub / Aalto Gitlab.

**Research background**

If some understanding of the scientific background wasn’t important, you might be hiring a software developer instead. Expect us to take some time to understand the science.
Understanding existing code

Also expect that, if there is any existing code, it will take some time to understand for a new person. Also, there is likely to be a period of refactoring to improve the existing code, where it seems like not much is getting done. This is a necessary step in investing for the future.

During the project

Our RSE will most likely want to go work with you, in your physical location (well, after corona-time), a lot of the time. It would be good to arrange a desk area as close as possible to existing researchers. “Mobile-space” but close is better than fixed but further.

Our goal isn’t just to provide a service, but to teach your group how to work better yourselves after the project.

Software quality and testing

Software which is untested can hardly be considered scientific. We will work with you to set up an automatic testing framework and other good practices so that you can ensure software quality, even after the project. This also ensures faster and more accurate development in the future. We’ll teach you how to maintain this going forward. This is in proportion to the complexity of the project and need.

We also pay particular attention to the maintenance burden of software: you’ll be using software much longer than you write it. We aim for simple, reliable strategies rather than the fanciest things right now.

After the project

We don’t want to drop support right after the project (that’s why you work with us, not an external software developer). Still, we have finite resources and can’t fund work on one project from another, so can’t do everything for everyone. You can expect us to try to passively keep supporting you for during the “daily garage” time as best we can.

If your department or unit provides basic funding (see the implementation plan), then long-term service is included, and this has no limits. However, this is shared among everyone in your unit, and focused on strategically support that helps many people.

Tracking scientific benefits

We need to record the benefits of this service:

- Researcher time saved
- Computer time saved
- Number of papers supported
- Software released or contributed to
- Open science outcomes (e.g. open software, data management)
- New work made possible (e.g. grant or project wouldn’t have been possible)
- Qualitative experience: increased satisfaction, educational outcomes, etc.
Releasing the software

A key goal of our support is releasing the software for broader use in the community (open science). Ideally, this will be a continual process (continue releasing as development goes forward), but we can prepare you for a first release later on, too.

We recognize the need to maintain a competitive advantage for your own work, but at the same time, if your work is not reproducible, it’s not science. We’ll work with you to find the right balance, but a common strategy is some core is open, while your actual analysis scripts which make use of that core are released with your articles.

Academic credit

Our RSEs do creative scientific work on your projects, which (depending on scope) can rise to the level of scientific authorship. This should be discussed early in the project.

- The software-based scientific creativity can be different than what is published in your articles: in this case, it can make sense to release the software separately.

- This is not to say that RSEs who work on a project should always be authors, but it should be considered at the start. See TENK guidelines on research integrity (authorship section).

- A contributing that is significant enough to become scientific novelty and such that the programmer must take responsibility for the outcome of the work usually rises to the level of co-authorship.

- It is OK to consider the code authorship as a separate output from the scientific ideas, and the RSE can help properly publish the code so that it is citeable separately from the paper.

See also

- UCL RSE group processes: That page heavily inspired this page. Broadly, most of what you read there also applies to us.

For group leaders

You, or someone in your group, has requested Research Software Engineer services for one of your group’s projects. This service provides specialist support for software, data, and open science so that you can focus on the science that is interesting to you. You probably have some questions about an outsider coming in to your project, and this page will answer those practical questions. For researchers using our services, also see How we work.

There are two funding strategies:

- Short term (a few days or less) might be funded by your department.

- Longer term is funded from your own projects.
Access to data

Our goal is not to come in, wave our hands, and leave you with something unusable. Instead, we want to come in and set you up to work yourself in the future. Thus, (if it’s necessary) we’ll want the same access to your group’s data/workspace/tools as you have.

This access is removed after the project is finished. We will try to remember this, but sometimes projects drag on with no clear ending (or you want long-term consultation), so you should also pay attention to this. Out of principle (+ policies), we don’t use admin access to access any of your data.

NDAs, intellectual property, etc.

The RSE staff are Aalto employees and are automatically bound to confidentiality, and have signed the same extra confidentiality agreement that Aalto IT system administrators have, and are similarly vetted.

Using our services doesn’t affect your intellectual property right any more than another employee working on the project will. This is service-for-pay, so you get all rights. However, our RSEs expect to be acknowledged according to good scientific practice (see How we work).

Funding

If the project lasts a short time (a few days or less), and you are in one of the sponsoring units, then the service may be free (depending on your unit’s policies).

Otherwise, you will be expected to fund the RSEs out of your own projects or basic funding. This is done by adding your project numbers to the RSE’s Halli, so that their salary can be directly paid from your project. The advantage is that this is identical to researcher funding, so it is compatible with almost all pre-existing grants.

We will contact you and your department’s controller to set the budget and get things set up. (If you are reading this, you are one of the pilot cases so we’ll figure it out together.)

New grants

If you are applying for a new grant, you can directly write Research Software Engineering services into your grant (or some similar name that sounds good to your reviewers). See For grant applicants.

For grant applicants

Warning: Grant applicants, if you are planning to use Aalto Research Software Engineers service, feel free to contact us at rse-group at aalto.fi, or fill out our request form.

This page is currently (2020-03) a draft for feedback. Please send corrections to the person that send this page to you, or rse-group at aalto.fi

If you’ve decided you would like to use the research software engineer services in your project for a long period, you might want to write it directly into your grant proposals. First off, your research will be better because you can you can assign porting/optimizing/scaling of codes, data management and open science, automation, and so on for the RSEs, while concentrating the main project resources on the actual research question. Second, it may make your proposal more attractive.
How it works

Short-term services are funded by various departments and schools and free to projects (part of the “research environment” services). Longer term service should be funded by projects - either an external grant or basic group fund. There are two ways to write this into a project proposal:

- As a purchased service, like usage of different infrastructures. This is flexible, but there is a limit of how much you can include in a grant. Note that indirect costs and overheads by the research site are not included, so these prices appear higher to cover that.
- As a research salary, just like other salaries on your project. This has fewer limits, but is less flexible (you get one person for a designated time period).

General grant considerations

You can find general boilerplate text to include in your proposals Boilerplate text for grant proposals, but you can read below to build it in even more.

Data Management / Open Science are big deals among some funders right now, and research engineers are perfect for helping with these things because they are experts in the associated technical challenges. The RSE service can help in the societal impact sections: your outputs will be more ready to be reused by society. You could, for example, promise deliver more types of outputs that aren’t exactly novel science but help society to use your results (e.g. databases, interactive visualisations, etc.).

Make sure you mention the general Science-IT infrastructure in the “research environment” section, i.e., the basic service provided by Aalto. You can copy something from the boilerplate text (first link in this section).

Academy of Finland

This applies to most general research grants, from the general terms and conditions. Funding may be used to cover costs related to the research plan or action plan. The research site must fund basic project facilities - which is the case at Aalto for basic RSE services.

Interesting terms from the Academy: it urges research data and methods to be freely available. 6.2.2: “Research data and material produced with Academy funding in research projects and research infrastructure projects must be made freely available as soon as possible after the research results have been published.”

- As a RSE salary:
  - “Salaries, fees and indirect employee costs” may be included in Academy projects. These may go to research software engineers, which to the academy appear equivalent to “normal” researchers. The RSEs are researchers.
  - Write in a Research Software Engineer as a salary for a set number of months. You may specify a name as N.N., or contact us for a name to include. We do not promise any one person at any one time, but we will work with you as much as possible. See LINK for costs per person. You can also contact us to discuss how much effort you may need.
  - The RSE and timeframe will be set at the beginning of the grant. It is possible to adjust later, but requires a significant amount of coordination between different services (thus the preference for the service purchase based approach, if it was possible).
  - Note that “We recommend that they be hired for a period of employment no shorter than the funding period, unless a shorter contract is necessary for special reasons dictated by the implementation of the research plan or action plan (or equivalent). Short-term research, studies or other assignments may also be carried out in the form of outsourced services.” So, consider the balance of this and purchased services.
• As a service purchase:

  – **Warning:** Our latest information indicates that internal billing (this service purchase) is not really possible for Academy grants. You must use “As a RSE salary” above.

  – The research site should provide “basic project facilities”, which Aalto does. Justify the extra purchase as beyond the basics.

  – Please see LINK for our general costs, and how many person-months you can get for a given price. Since estimating the amount of effort needed is difficult, [contact us](#) and we can help you prepare an estimate.

  – Maximum amount: We recommend you include no more than XXXXX as a service purchase. Please see LINK (login required) for our prices, when paid via external funding.

  – Justification for funding (include in proposal): “Technical specialist work to ensure scientific and societal impact outputs follow best practices in software development and research data management practices, so that they can be of greatest possible benefit to society.”

  – Flexibility: we could flexibly invoice as needed for your project. You don’t have to decide the time period in advance (only follow your submitted budget), and different RSEs can work on different parts of the problem, so you always have the best person for the job.

**European Commission grants**

<table>
<thead>
<tr>
<th>Warning:</th>
<th>Internal billing is (for practical purposes) not possible for EC grants. Use RSE salary as for Academy of Finland.</th>
</tr>
</thead>
</table>

**7.1.3 About research software engineers**

**RSE community**

Do you like coding, research, and the academic environment, but want slightly more emphasis and community around the software side? Join the Aalto RSE community. You can join whatever your current position is, you don’t need to be hired as a research software engineer. There are no requirements, just networking and development. This is also a “Triton powerusers group”.

RSEs have been an essential part of science for ages, but are hardly ever recognised. We have many here at Aalto. Aalto SciComp is trying to make a community of these people. By taking part, you can:

  • Network with others in similar situations and discover career opportunities.
  
  • Share knowledge among ourselves (maybe have relevant training, too).
  
  • Take part in developing our services - basically, be a voice of the users.
  
  • More directly help the community by, for example, [directly updating this site](#), helping to develop services, or teaching with us.

To join the community, see the [general SciComp community page](#). Also, you should take part in the Nordic-RSE Finland chats - there is a strong Aalto presence there, and we use that as our Aalto chat time, too.
For RSE candidates and community

This page is designed to guide people into the interesting world of research software engineering: providing a view into the types of skills that are useful to research groups at Aalto. It also provides links to training material which our RSEs should learn after starting (so don’t be intimidated by what you see on here!).

Do you like coding and research, but don’t want an academic career path with publications as your sole purpose? Be a Research Software Engineer with us! Our pilot is designed to bridge the gap between academic research and a future career in research software or a research scientist in a company.

If (some of) the following apply to you, you are a good candidate:

- I like the academic environment, but don’t want to focus just on making publications.
- I am reasonably good at some programming concepts, and am eager to learn more. I know one language well, can shell script, and generally familiar with Linux.
- I am interested in going to a scientist-developer kind of role in a company, but need more experience before I can make the transition.

General qualifications and duties

We strongly prefer good computational researchers (PhD level preferred) who can improve their software development skills than the other way around. This role can be combined with other roles, but note that this is not targeted to those who intend to follow a tenure-track academic career path. New RSEs will get a training period which rounds out any missing skills. They will also be involved in a complete support package: they will have the chance to be involved in teaching an infrastructure development.

Job advertisements

From time to time, job advertisements are posted on the Aalto University job portal, with notices on various other channels including Aalto scientific computing mailing lists.

What you do

At least at Aalto, you will:

- Provide software development and consulting as a service, depending on demand from research groups.
- Provide one-on-one research support from a software, programming, Linux, data, and infrastructure perspective: short-term projects helping researchers with specific tasks, so that the researchers gain competence to work independently.
- As needed and desired, teach and provide other research support.
- A typical cycle involves evaluating potential client researchers, meeting, formulating a work plan, co-working to develop a solution, teaching and mentoring for skill development, and follow-up.

All will be done as part of a team to round out skills and continuous internal knowledge-sharing.
Skillset

Below, we have a large list of the types of technologies which are valued by our researchers and useful to our RSEs. No one person is expected to know everything, but we will hire a variety of people to cover many of the things you see here.

Most important is do you want to learn things from this list? Can you do so mostly independently but with the help of a great team?

General tech skills

Our broad background on which we build:

- Basic mandatory skills include Linux, shell scripting, some low-level programming language (C, Fortran), and programming in several more languages (Python particularly advantageous).
- Good knowledge of computer clusters, batch systems, and high-performance computing.
- Any additional programming, workflow, research, or system tools are a plus. You should have a wide range of skills, but the exact skills are not so important. Most important is sufficient fluency to pick up anything quickly. These skills should be listed as an appendix to the cover letter if not included in the CV.
- Advanced parallel programming skills are a plus, but equally important is the ability to create good, simple, practical tools.
- Git, GitHub, git-based collaborative workflows.
- Software testing, CI, documentation, reproducible, portability, etc.
- As an example, the ideal candidate will have near-perfect knowledge of all Software Carpentry, CodeRefinery, and the generic parts of our HPC lessons - or be able to fill in gaps with minimal effort.
- But at the same time, we don’t just want people from purely computational backgrounds. You’ll work with people from experimental sciences, digital humanities, etc, and good people from these backgrounds are important, too.
- A good attitude towards mentoring and teaching and an ability to explain complex subjects in an accessible way.
- Commitment to diversity and equality of researchers among many different backgrounds.
- Good knowledge of English. Finnish is advantageous but not required, our internal working language is English.

Teaching and mentoring skills

You won’t just preforming technical tasks. As part of being a RSE, you need to help others to be self-sufficient as well. This requires teaching and mentoring skills.

- How to help someone use a computer by Phil Agre
- Motivation and demotivation, a chapter in Teaching Tech Together.
- History of the RSE concept
Specific examples

This is a selection of advanced skills which are useful (remember, this is what you might learn, not what you already know):

- Advanced experience of debugging/profiling/developing Linux tools, including Git, Intel and GNU compiler suits and corresponding tools.
- Software building tools like Make, CMake and alike.
- Advanced knowledge of parallel programming models, experience of parallel programming (OpenMP, MPI).
- Advanced GPU computing / programming (CUDA, OpenACC, OpenMP models), experience of porting software to GPUs.
- Profiling and optimization - both of low-level languages and high-level.
- Knowledge of scientific software and packages including Matlab, Mathematica, Python libs, others is beneficial.
- Experimental data collection, LabView, etc.
- Workflow automation, shell scripting, porting from single machines to clusters.
- Docker, Singularity, containers.
- Data analysis tools like R, Python, pandas, numpy, etc. are beneficial.
- Julia, Matlab, Mathematica.
- Web development, cloud operations.
- Scientific Computing on other operating systems.

Open science and data

As a RSE, you should also serve as an advocate for open science, reproducible research, and data management.

- Data management, data engineering, data wrangling.
- Reproducible research.
- Open source software development, community formation.
- Software packaging and distribution, (e.g. PyPI, conda, etc.).

Training resources

These resources may be interesting to support your career as an RSE:

- Hands-on scientific computing
- Software Carpentry
- CodeRefinery
- Nordic-RSE
- The Society of Research Software Engineering
- HPC and Triton
7.1.4 Checklists

Python project checklist

This checklist covers major considerations when creating a high-quality, maintainable, reusable Python codebase. It is designed to be used along with a RSE to guide you through it (it is in a draft stage, and doesn’t have link to what these mean). Not everything is expected for every project, but a sufficiently advanced complicated project will have most of these things.

- Citeability and credit, authorship discussion
- License
- Version control
  - In use locally
  - In use on some platform (Github/Gitlab/etc)
  - Regular commits
  - Discuss issue tracker
  - Make one example pull request
- Modular design
  - Standard project layout
  - Importable modules
  - Command line or other standard interface
  - (relates to packaging below)
- Tests
  - Recommendation: pytest
  - Simple system tests on basic examples
  - More fine-grained integration or unit tests
  - CI setup
  - Test coverage
- Documentation
  - Forms / levels
    * README file: good enough?
    * Project webpage
    * Sphinx project
    * Read The Docs
  - To include
    * About
    * Installation
    * Tutorials
    * How to / simple examples to copy
7.1.5 Internal documents

We believe in openness, so make our procedures open. They are subject to improvement at any time.

Time tracking

**Warning:** This page is still in draft form and being discussed and developed - it is only a proposal. See the note on the parent page.

This proposal may turn out to be especially bad... please comment.

Unfortunately (fortunately?), we have to track our time some, in order to justify the benefits of what we do.

Finance time tracking

For projects funded by groups (external or internal funding), they should me marked in Halli. All other projects (funded by the department’s/school’s basic funding) is marked to the standard RSE project (ask for it), and this time is accounted for at the end of each year (using the system below).

Internal time tracking

In addition to the financial tracking above, it seems we have to keep a separate tracking of what projects we work on because not every project is reportable via Halli.

Right now we propose that time tracking is done through Gitlab, within the issue opened for each “project”.

Gitlab commands:

- Use these within the issue as a comment, to control the time allocation.
- `/estimate NNw` - estimate total time a project make take. Used as soon as possible at beginning of a project, can always be updated
- `/spend NNh` - announce that you have spent a certain amount of time on the project
- Units: Months (mo), Weeks (w), Days (d), Hours (h), Minutes (m). Default conversion rates are 1mo = 4w, 1w = 5d, and 1d = 8h.
- Use the labels to record the sponsoring department, funding source (project or basic), and state.
Aalto scientific computing guide

Reporting

RSEs should be able to produce tabular data matching this semantic model. Each row should be one (project, day) work report.

- **username** of the RSE
- **unit** hosting the research (Aalto acronym: SCI, (CS, NBE, PHYS, MS, DIEM), ARTS, BIZ, CHEM, ELEC, ENG). This can be found from Gitlab (inferred from **project-id**).
- **day** of work (YYYY-MM-DD)
- **hours** of work on that day.
- **funding**: **project** or **basic** funding - who is paying for this project? If **project**, this implies that it was billed to Halli. If **basic**, it's assumed that it was billed to the RSE project and accounting will be done at the end of the year. Note that this can’t be gotten straight from Gitlab, since **project**-funded project will usually also have some **basic**-funded consultations in addition.
- **project-id**: Issue number from Gitlab (optional? - or some other ID?).
- **comment**

Things do not have to be exact for every day, but when aggregated over months, it should asymptotically approach the right values.

Other notes

Be aware:

- Be aware that it takes some time to get up to speed with a project. This should be considered when making the initial estimate, during the first consultation.
- When being paid by projects, we need to only record time actually spent on that project. Thus, daily garages and other RSE meetings need to recorded to the common RSE project/cost center. These overhead work times are managed separately.

Implementation

About this page

This is our tentative implementation plan, as of August 2020. It is always subject to revision, but is a somewhat controlled document.

About

Researcher Software Engineers provide specialized scientific computing and data management support to researchers, beyond what is currently offered by Science-IT. Their funding is guaranteed by departments/schools/other units, but after the ramp-up phase most funding is expected to come from the research projects themselves.

Services include, for example, software development, scaling up or optimizing computations, taking new technologies into use, and in general promoting best practices in new and existing research using computational methods.
Funding has three types:

- **Ramp-up/Guarantee (R/G):** Ramp-up funding to do initial hires, until project funding takes over
  - Ramp-up: department/schools/other units allocate a certain amount of money to do hires.
  - Units which provide Ramp-up/guarantee get first priority for their projects.
  - Replaced with project funding (below), if there are no projects then used for basic services (below).

- **Project (P):** External or group money, allocated by a PI for a specific task in their group.

- **Basic (B):** Allocated from units for short-term basic service for all of its members.
  - Allows short, strategic assistance without microtransactions
  - Science-IT work is a type of basic work, but may be requested by the Science-IT team instead of the researchers themselves. (For example, Science-IT has a long list of inefficient hardware use and inefficient software practices which can keep RSEs occupied for a long time. RSEs can also work on Triton/scientific computing technical development projects, which helps RSEs gain competence for the rest of their tasks.)

### Time allocation principles

- We track time spent per unit. Fairshare algorithm: the unit with the largest “deficit” in time gets priority for upcoming projects.
- Units which provide ramp-up/guarantee funding get priority for their projects.
- Project funding replaces ramp-up/guarantee funding.
- Time paid from basic funding is allocated to tasks within the unit with the greatest strategic benefit, for example helping an entire group to use better tools or fixing extreme waste of resources.
- When a group provides project funding, they can decide the tasks the RSE will do.

### Ramp-up plan

This is a rough estimate of the type of demand we expect:

<table>
<thead>
<tr>
<th>Distribution of work</th>
<th>2020 H2</th>
<th>2021</th>
<th>2022</th>
<th>2023</th>
<th>Long-term</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTE</td>
<td>2</td>
<td>2–3</td>
<td>3–4</td>
<td>3–5</td>
<td>4+</td>
</tr>
<tr>
<td>Project work</td>
<td>20%</td>
<td>50%</td>
<td>60%</td>
<td>70%</td>
<td>70%</td>
</tr>
<tr>
<td>Basic work for units</td>
<td>50%</td>
<td>40%</td>
<td>30%</td>
<td>20%</td>
<td>20%</td>
</tr>
<tr>
<td>Basic work for Science-IT</td>
<td>30%</td>
<td>10%</td>
<td>10%</td>
<td>10%</td>
<td>10%</td>
</tr>
</tbody>
</table>

- Our initial survey reached only Triton users and had 40 responses, of them 60% said “quick consultation”, 60% said “short term, 1–2 days”, 40% said “medium term, weeks to months”.
- Actual ramp-up depends on funding cycles, research timing, and human psychology.
Start-up funding (already guaranteed)

(section removed; to be placed elsewhere)

Funding practicalities

Principle: the daily rate is roughly equal to “senior postdoc/staff scientist” salary + overheads.
Principle: When working for a research project, the RSEs record those working hours in Halli to that project. The corresponding portion of the salary is then automatically charged to the project. Remaining hours are recorded to the Dean’s unit RSE project, and once a year we split these costs and send them to each department. [Updated 2020-11-05]
(details to be filled in by Finance)

Measurement and KPIs

- Number of projects and researchers who have been given support
- Number of researcher-days saved, as estimated by our customers.
- Fraction of project funding vs total program funding

Communication

- Units which fund us will be informed of our activities at least every 6 months.
- “As open as possible, as closed as necessary”. All RSE program data, documents, and statistics will be public, excluding actual project funding and information from the customers.

Risks and ramp-down

- Primary risk: making permanent hires, yet not being able to sustain the program long-term.
  - Mitigation: we will only hire RSEs which can be absorbed into Science-IT naturally should the need of this service fade away.
- Risk: difficulty in reaching researchers and explaining what we do
  - Mitigation: Science-IT has a long list of researchers who are using research services inefficiently: they can be contacted directly to inform about this service. Helping them and producing best practice examples for the future can keep several people busy for years.
- Risk: Researchers see need, but group leaders unwilling to pay
  - This is indeed a risk, but there is precedence from other countries that there are enough people willing to pay. There will likely be a slow start, but as time goes on, expenses incurred by this service can directly be written into the budget of funding applications.

In our ramp-down strategy, we absorb the RSEs into Science-IT, CS-IT as part of its development efforts, or into other existing teams.
Job descriptions

Warning: This page is still in draft form and being discussed and developed. See the note on the parent page.

These are job descriptions for RSE descriptions. They are not yet formal HR job descriptions and won’t be directly used as such, but provide a vision of our career ladder.

A RSE is researcher whose advancement of science is not defined by number of papers, but by quality of software and contributions to open science.

RSE 1

A RSE1 is just starting their career and is being introduced both to software tools and the research process. This RSE would get mentoring much like a new doctoral student does, but instead of aiming to publications, they would aim to quality, released software.

Qualifications: Masters degree, thesis in combining computation and research or software development with some research qualifications, but little real-world research experience.

Pay/job level: roughly like master’s employee or PhD student. Advancement: would be expected within 1-2 years.

RSE 2

Able to competently work on own projects using tools they know while learning new tools effortlessly. They are currently learning to finding the right tool for the job and to connect the technical task (software and data related) to the impact to society, Aalto, and individual grants.

This is roughly equivalent to a postdoctoral researcher, a transition time between academic skills of a doctorate and whatever may come next. In particular, this can serve as a bridge between a (somewhat more theoretically focused) doctorate degree and a job in industry, and CV and skills development is in line with this.

Qualifications: Doctorate or extended work experience. Pay/job level: similar to postdoc.

Advancement: expected to advance within 2-3 years. This person is still in training (much like a postdoc) and is probably deciding which way to take their career.

RSE 3

Like above, but is additionally able to independently negotiate with research groups to plan a project, including deciding tools and expected results. In particular, a RSE 3 should be able to explain the value of good software practices to the researchers and plan/advocate for good open science and research data management practices across various fields.

Pay/job level: like staff scientist, always permanent.

Advancement: A person is a competent, independent scientist/engineer at this point, and advancing is not needed for everyone. Of course, lifelong learning always continues. To be honest, advancing in the academic system is difficult, and many people will make a horizontal move to another place.
Beyond

Further career development is not a part of the Aalto RSE program (yet?), and to be honest it’s hard to see an internal advancement in the current academic system. Still, there are many ways people can continue their career development depending on their career goals, for example:

- Tech lead of larger RSE projects
- Study and develop new technologies for production
- Management, either of RSE group or other services
- Applying for grants, leading projects, etc. as a staff scientist might do
- Mentoring or supervising students or other researchers

At Aalto, these aspects are not yet developed, and some of them would be horizontal moves outside the RSE team (or collaboration with someone outside the team). At some point, people have to take their careers in the direction they want and begin combining various unique skills.

Commercial developers

We don’t plan on competing with commercial developers, but the difference with a RSE3 is that:

- A software developer can do what is asked, but not work with the researcher to figure out what they actually need. The software developer will probably be slightly more requirements-product based, rather than agile-research work to develop a tool over time.
- A software developer make produce a product that is not sustainable in an academic setting: requires too much focus and specialized knowledge to be improved in an academic environment.
- A software developer may use more modern and industrial-scale tools.
- A software developer from outside would come in and leave, a RSE in this group would provide longer-term support (but this is more a property of the group, not the person).

Unit information

This page describes the Aalto units which are supporting the RSE program and what their priorities are.

General questions

A unit joining and providing funding should first check the implementation plan and then consider the following questions:

- Maintain some fraction of time for basic services (short term, free-at-point-of-use assistance to strategically improve the quality of your research) or prioritize projects that can pay themselves to make the unit’s cost as low as possible.
- Support the coolest projects help those who are struggling the most.
- Support efficient use of infrastructure/hardware or existing project new projects.
- Are there any strategic areas to focus on?
- Contact us for a personalized chat about possibilities.
Units

The following units are currently supporting the RSE program, so only these units may receive the “basic service” (short term help, free to the user). Others may be able to purchase our services using their own project funding, depending on how much time we have available.

SCI
CS
NBE
PHYS

Advisory board

Warning: This page is a draft.

This page describes the advisory board of the Aalto RSE program and hosts the results of its meetings. Out of principle, all material is open on this page (though specific items may be retracted).

Purpose of the advisory board

The advisory board provides advice to the strategy (and when relevant, day to day implementation) of the Aalto RSE program and its relation to research, scientific computing, and teaching at Aalto.

Current advisory board

• ...
• ...
• ...
• ...
• ...
• ...
• ...
• ...
• ...
• ...
Meetings

Topics for the next meeting and results from previous meetings are located here, newest first.

Next meeting

• Purpose of advisory board and its roles. How often to meet?
• What are your priorities?
  • What is the threshold for your department to “pay” for service.
• How can we find customers?
• How much do we focus on cost recovery, and how much on basic work?
• What are our KPIs? See *Measurement and KPIs* and *Tracking scientific benefits*.
  – Cost recovery from projects
  – N ongoing projects and N completed projects
  – N publications supported.
  – N open outputs produced (non-publication: datasets, software, etc.)
  – Survey (of PIs) of benefits after.
  – Estimated time saved.

7.1.6 See also

• The *Nvidia AI Tech Center* provides free RSE services for research projects for Finnish Center for AI members (includes Aalto).

Related programs from Aalto Scientific Computing

This is the hands-on compliment our training programs, mainly *scientific computing in practice sessions* and CodeRefinery. If you want a kickstart to put those sessions in practice, this is for you. Attendance in these courses is useful but by no means required.

Other links

• *Aalto Scientific Computing*, the organization behind this program.
• Nordic RSE community, currently in the process of being formed (Aalto SciComp and the RSE program is a member).
• History of the RSEs concept
• Who is a RSE?
• Keynote video by Mike Croucher on the rise of RSEs and their benefits
• The UK RSE association is quite advanced in promoting RSE careers.
• RSE international.
• Note the bottom section on page 105(print)/106(PDF) of the 2018 Research, Art, and Impact assessment.
In this section, you find general (not Aalto specific) scientific computing resources.

8.1 Scientific computing tips

8.1.1 Encryption for researchers

This page describes the basics of encryption to an audience of researchers. It describes how it may be useful (and when not needed) in a professional researcher environment, in order to secure data. It doesn’t describe encryption for personal use (there are plenty of other guides for that). It doesn’t go into very deep details about cryptography. It doesn’t get go into deep details. Cryptography is the type of things where there are a huge number of subtle details, and everyone has their own opinion. This guide is designed to provide an understanding for basic use, not cover every single subtle point.

Status: this is somewhat complete, but is not a complete guide. It will be extended as needed.

Summary

Modern cryptography is quite well developed, and available many places. However, the human side is very difficult. Encrypting something, but not keeping the key or password secure, has no benefits. To use your encryption, you need to decide what your goals are (who should access, who you want to keep safe from) and then plan accordingly. The security of cryptography is decided more by how you manage the keys and process than the deepest technical details.

Key management

The point of encryption is to trade a hard problem (keeping a lot of data secure) to a more limited problem (keeping a single key or password secure). These keys can be managed separately from the data. This immediately takes us to the importance of key management. Let’s say you can’t send data over email unless it is encrypted. If you encrypt it and send the password in the same email as the encrypted data, you have managed to technically satisfy the requirement while adding no real security at all. A better strategy would be to give the password to someone when you meet them in person, send it by another channel (e.g. SMS, but then it is only secure as SMS+email), or even better use asymmetric encryption (see below).

Deciding how you will manage keys is the hardest part of using encryption. For some fun, next time you hear someone talk about using encryption, see if they mention how they keep the keys secure. Usually, the don’t, and you have no way of knowing if they actually are doing it securely.
Symmetric vs asymmetric encryption

There are two main types of cryptography. They can both be considered equally secure, but have different ways of managing keys.

Symmetric encryption uses the same password/key for encrypting and decrypting. It is good because it is simple, because there is only one key or password you need to know and it is easy to think “one data=one password”. However, everyone needs to know the same password, and it can’t be changed. Since the same password has to be everywhere, this can be a bit insecure depending on the use, and you can argue it’s a bit complicated to keep that key password secure (if there are many people, or if it needs to be automated).

Asymmetric encryption has different keys for encrypting and decrypting. So, you use a “public key” to do an encryption (which requires no password - everyone in the world can know this key and your data is still secure). You have a separate private key (+password) which allows only you to decrypt it. This separation of encryption and decryption was a major mathematical breakthrough. Then, anyone who needed to receive data securely would have their own public/private key, and all the public keys are, well, public. When you want to send data to someone, you just encrypt it using their public key, and there is no need to manage sharing a password. This allows you to: encrypt so that multiple people can read it, encrypt automatically without password, and encrypt to someone not involved in the initial process.

With asymmetric encryption, there are some more things to consider. How do you make sure that you have the right public key?

Encryption programs

This lists some common programs, but this should not be taken to mean that using these programs makes your data safe. Security depends no how you use the program, and security will only decrease over time as new analysis is done. It is usually best to choose well-supported open source programs where possible. More detailed instructions will be provided as needed.

7zip

7zip is a file archiver (like zip). It can symmetrically encrypt files with a passphrase.

PGP

PGP is a set of encryption standards (and also a program). It has a full suite of encryption tools, and is quite stable and well-supported. You often hear about PGP in the context of email encryption, but it can be used for many things. On Linux systems, it is normally found as the program gpg (Gnu Privacy Guard). This guide uses gpg.

Full disk encryption

Programs can encrypt the entire hard disk of your computer. This means that any data on it is safe, should your computer be lost. There are programs to do this for every operating system, and Aalto laptops now come encrypted by default.
Using symmetric encryption with gpg

Encryption:

```
gpg --symmetric input.file
```

Decryption:

```
gpg input.file.gpg
```

This will ask you for a password. If you do not want it to, you can use `--passphrase-fd` to pass it automatically. Normally, keeping a password in a file is considered quite insecure! Make sure that the permissions are restrictive. Anyone that can read this file once be able to read your data forever. The file could be backed up and spread all over the place - is that what you want? IT admins will be technically able to see the passphrase (though they do not). Is this all within the scope of your requirements?

```
cat pass.txt | gpg --passphrase-fd 0 --symmetric input.file
```

Using asymmetric encryption with gpg

When using asymmetric (public key) encryption, you have to generate two keys: public and private (they are made at the same time). The private key must be kept private, and has a passphrase on it too. This provides an added level of security on top of the file permissions.

There are plenty of guides on this available. Some examples:

- https://www.madboa.com/geek/gpg-quickstart/
- https://gnupg.org/documentation/index.html

You can encrypt a single files to multiple keys. This means that the owner of any of the private keys can decrypt the file. This can be useful for backups and disaster recovery.

General warnings

- Strong encryption is serious business. It is designed so that no one can read the data should the keys or passwords be lost. If you mess this up and lose the key/password, your data is gone forever. You must have backups (and those backups must also be secure), …
- If you keep passwords in files, or send them insecurely anyhow, then the technical security of your data is only as great as of that key/password.
- The strength of your encryption also depends on the strength of your password (there is the reason it is often called a “passphrase” - a phrase is more secure than a standard password). Choose it carefully.

Advanced / to do

- How much security is enough?
- Set cipher to AES (pre 16.04)
8.1.2 Git

Git is a version control system. This page collects various Git tutorials and resources

Version control systems track changes to files. This means that as you are working on your projects (code, LaTeX, notes, etc), you can track history. This means that you can see former history, and collaborate better. Using one for at least for code should probably be one of the minimum standards of computational research.

“Git is a distributed version control system designed to handle everything from small to very large projects with speed and efficiency. Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.”

Note:

- This page is git in general, not Aalto specific.
- aalto/git contains advice on the Aalto Gitlab, a repository for the Aalto community integrated to Aalto systems.

Basic git tutorials

- There is an interactive git tutorial from codeschool and github. Good for your first use.
- Software carpentry has a good tutorial focused on researchers.
- Gitlab cheatsheet.

More references

- You can search for many tutorials online.
- software-carpentry.org (an organization that teaches development to scientists) has a very good tutorial online.
- The book “Pro Git” is online.
  - Read chapters 1-3 for a good introduction to using git for your own projects.
  - Read chapter 5 for a good introduction to using git to collaborate with others.
- There’s a somewhat official documentation place - including videos.
- There is an official tutorial but it is probably too theoretical.
- All git commands have very good but very detailed manual pages - type man git COMMAND or git help COMMAND to see them.
- Interactive git cheatsheet. (very good once you know the basics)
- A Visual Git Reference

Gitlab-specific information:

- A tutorial
Other hosting services

Realistically, use version.aalto.fi for most projects related to Aalto research, and Github if you want to make something open-source with a wider community (but you can also make open repos in Aalto Gitlab, just harder for random people to contribute). For non-work private repos, you have to make your own choice.

- Github is a proprietary commercial service, but extremely popular. No free private repositories or groups (but you can pay).
- Bitbucket is also somewhat popular, limit of free 5 private repositories (but you can pay for more).
- Gitlab.com is a commercial service but makes the open-source Gitlab edition. Gitlab.com offers unlimited private repositories.
- source.coderefinery.org is another Gitlab hosted by the Coderefinery project, a pan-Nordic academic group. It might be useful if you have a very distributed project, but realistically for Aalto projects, use Aalto gitlab.

8.1.3 Pitfalls of Jupyter Notebooks

Jupyter Notebooks are a great tool for research, data science type things, and teaching. But they are not perfect - they support exploration, but not other parts of the coding phase such as modularity and scaling. This page lists some common limitations and pitfalls and what you can do to avoid them.

**Do** use notebooks if you like, but **do** keep in mind their limitations, how to avoid them, and you can get the best of both worlds.

None of the limitations on this page are specific to notebooks - in fact we’ve seen most of them in scripts long before notebooks were popular.

**Modularity**

We all agree that code modularity is important - but Jupyter encourages you to put most code directly into cells so that you can best use interactive tools. But to make code the most modular, you want lots of functions, classes, etc. Put another way, the most modular code has nothing except function/class/variable/import definitions touching the left margin - but in Jupyter, almost everything touches the left margin.

Solutions:

- Slowly work towards functions/classes/etc where appropriate, but realize it’s not as easy to inspect their insides as non-function code.
- Be aware of the transition to modules - do it when you need to. See the next point.
- Try to plan so it’s not too painful to make the conversion when the time comes.

**Transitioning to modules**

You may start coding in notebooks, but once your project gets larger, you will need to start using your code more places. Do you copy and paste? At this point, you will want to split your core code into regular Python modules, import them into your notebooks, and use the notebooks as an interface to them - so that modules are somewhat standard working code and notebooks are the exploration and interactive layer. But when does that happen? It is difficult to make that transition unless you really try hard, because it’s easier to just keep on going.

Solutions:

- Remember that you will probably need to form a proper module eventually. Plan for it and do it quickly once you need to.
• Make sure you notebooks aren’t disconnected from your own Python code in modules/packages.

• You can set modules to automatically reload with \texttt{%load_ext autoreload, %autoreload 1}, and then \texttt{%aimport module_name}. Then your edits to the Python source code are immediately used without restarting and your work is not slowed down much. See more at the IPython docs on autoreload (note: this is Python kernel specific).

• \texttt{importnb} to import notebooks as modules - but maybe if you get to this, you need to rethink your goal.

**Difficulty to test**

For the same reasons modularity outlined above, it’s hard to test notebooks using the traditional unit testing means (if you can’t import notebooks into other modules, you can’t do much). Testing is important to ensure the accuracy of code.

Solution: Include mini-tests / assertions liberally. Split to modules when it is necessary - maybe you only create a proper testing system once you transition to modules.

Solutions:

• Various extensions to pytest that work with notebooks
  – \texttt{nbval, pytest-notebook}: run notebook, check actual outputs match outputs in ipynb.
  – \texttt{pytest-ipynb}: cells are unit tests
  – This list isn’t complete or a recommendation

• But just like with modularity above, a notebook designed to be easily testable isn’t designed for interactive work.

• Transition to modules instead of testing in the notebook.

**Version control**

Notebooks can’t be version controlled well, since they are JSON format. Of course, they can be version controlled (and should be), and there are a variety of good solutions so this shouldn’t stop you.

Solutions:

• Don’t let this stop you. Do version control your notebooks (and don’t forget to commit often!), even if you don’t use any of the other strategies.

• \texttt{nbdime} - diffing and merging, VCS integration

• Jupyter lab / notebook git integration work well.

• Notebooks in other plain-text formats: Rmarkdown, Jupyter text (pair notebooks with plain text versions).

• Remember, blobs in version control is still better than nothing.
Hidden state is opposed to reproducibility

This is a bit of an obscure one: people always say that notebooks are good for reproducibility. But they also allow you to run cells in different orders, delete cells after it has run, change code after you run it, and so on. And this is the whole point of notebooks. So it’s very easy to get into a state where you have variables defined which aren’t in your current code and you don’t remember how you got them. Since old output is saved, you might not realize this until it’s too late.

Solutions:

- Use “Restart and run all” liberally. Unless you do, you can’t be sure that your code will reproduce your output.
- But wait… part of the point of notebooks is that you can keep data in memory instead of recalculating each time you run. “Restart and run all” defeats the purpose of that, so… balance it out.)
- Design for modularity and clean interfaces, even within a notebook. Don’t make a mess.

Notebooks aren’t named by default

This is really small, but notebooks aren’t named by default. If you don’t name them well, you will end up with a big mess. Also somewhat related, notebooks tend to purpose drift: they start for one thing then end up with a lot of random stuff in them. How do you find what you need? Obviously this isn’t specific to notebooks, but the interactive nature and modularity-second makes the problem more visible.

Solutions:

- Remember to name notebooks wells, immediately after making them.
- Keep mind of when they start to feature drift too much, or have too many unrelated things in them. Take some time to sort your code logically once that happens.

Difficult to integrate into other execution systems

A notebook is designed for interactive use - you can run them from the command line with various commands. But there’s no good command line interface to pass arguments, input and output, and so on. So you write one notebook, but can’t easily turn it into a flexible script to be used many times.

Solutions:

- Modularize your code and notebooks. Use notebooks to explore, scripts to run in bulk.
- Create command line interfaces to your libraries, use that instead of notebooks.
- There are many different tools to parameterize and execute notebooks, if you think you can keep stuff organized:
  - nbconvert
  - papermill
  - nbscript (self-advertisement)
  - … and plenty more
Jupyter disconnected from other computing

This is also a philosophical one: some Jupyter systems are designed to insulate the user from the complexities of the operating system. When someone needs to go beyond Jupyter to other forms of computing (such as ssh on cluster), are they prepared?

Solutions:

- This is more of a mindset than anything else.
- System designers should not go through extra efforts to hide the underlying operating system, nor separate the Jupyter systems from other systems.
- Include non-Jupyter training, some intro to the shell, etc. in the Jupyter user training.

Summary

The notebooks can be great for starting projects and interactive exploration. However, as a project gets more advanced, you will eventually find that the linear nature of notebooks is a limitation because code can not really be reused. It is possible to define functions/classes within the notebook, but you lose the power of inspection (they are just seen as single blocks) and can’t share code across notebooks (and copy and paste is bad). This doesn’t mean to not use notebooks: but do keep this in mind, and once your methods are mature enough (you are using the same code in multiple places), try to move the core functions and classes out into a separate library, and import this into the day-to-day exploration notebooks. For more about problems with notebooks and how to avoid them, see this fun talk “I don’t like notebooks” by Joel Grus. These problems are not specific to notebooks, and will make your science better.

In a cluster environment, notebooks are inefficient for big calculations because you must reserve your resources in advance, but most of the time the notebooks are not using all their resources. Instead, use notebooks for exploration and light calculation. When you need to scale up and run on the cluster, separate the calculation from the exploration. Best is to create actual programs (start, run, end, non-interactive) and submit those to the queue. Use notebooks to explore and process the output. A general rule of thumb is “if you would be upset that your notebook restarted, it’s time to split out the calculation”.

Notebooks are hard to version control, so you should look at the Jupyter diff and merge tools. Just because notebooks is interactive doesn’t mean version control is any less important! The “split core functions into a library” is also related: that library should be in version control at least.

Don’t open the same notebook more than once at the same time - you will get conflicts.

References

- This funny talk “I don’t like notebooks” by Joel Grus provided a starting point of this list.

8.1.4 nbscript: run notebooks as scripts

Warning: This page and nbscript are under active development.

Notebooks as scripts?

Jupyter is good for interactive work and exploration, but eventually you need more resources than an interactive session can provide. nbscript is a tool (written by us) that lets you run Jupyter notebooks just like you would Python files. (nbscript main site)
See also:

Other tools: There are other tools that run notebooks non-interactively, but (in my opinion) they treat command-line execution as an afterthought. There is a long-standing standard for running scripts on UNIX-like systems, and if you don’t use that, you are staying locked in to Jupyter stuff: the two worlds should be connected seamlessly. Links to more tools here.

Once you start running notebooks as scripts, you really need to think about how modular your whole workflow is. Mainly, think about dividing your work into separate preprocessing (“easy”), analysis (“takes lots of time and memory”), and visualization/post processing (“easy”) stages. Only the analysis phase needs to be run non-interactively at first (to take advantage of more resources or parallelize), but other parts can still be done interactively through Jupyter. You also need to design the analysis part so that it can run on a small amount of data for development and debugging, and the whole data for the actual processing. You can read more general advice at Jupyter notebook pitfalls.

Concrete examples include:

- Run your notebook efficiently on a separate machine with GPUs.
- Run your code in parallel with many more processors
- Run your code as a Slurm batch job or array job, specifying exactly the resources you need.

**nbscript basics**

The idea is nbscript input.ipynb has exactly the same kind of interface you expect from bash input.sh or python input.py: command line arguments (including input files), printing to standard output. Since notebooks don’t normally have any of these concepts and you probably still want to run the notebook through the Jupyter interface, there is a delicate balance.

Basic usage from command line. To access these command line arguments, see the next section:

```
$ nbscript input.ipynb [argument1] [argument2]
```

If you want to save the output automatically, and not have it printed to standard output:

```
$ nbscript --save input.ipynb                      # saves to input.out.ipynb
$ nbscript --save --timestamp input.ipynb        # saves to input.out.TIMESTAMP.ipynb
```

If you want to submit to a cluster using Slurm, you can do that with snotebook. These all run automatically with --save --timestamp to save the output:

```
$ snotebook --mem=5G --time=1-12:00 input.ipynb
```

**Setting up your notebook**

You need to carefully design your notebook if you want it to be usable both as a script and as through Jupyter. This section gives some common patterns you may want to use.

Detect if your notebook is running via nbscript, or not:

```
import nbscript
if nbscript.argv is not None:
    # We *are* running through nbscript
```

Get the command line arguments through nbscript. This is None if you are not running through nbscript:

```
import nbscript
nbscript.argv
```
Aalto scientific computing guide

You can use argparse like normal to parse arguments when non-interactive (take argv from above):

```python
import argparse
parser = argparse.ArgumentParser()
parser.add_argument('input', help='Input file')
args = parser.parse_args(args=argv)
```

Save some variables or save file if not running through nbscript:

```python
if nbscript.argv is not None:
    import cPickle as pickle
    state = dict(results=some_array,
                  other_results=other_array,
                  )
    pickle.dump(state, open('variables.pickle'), pickle.HIGHEST_PROTOCOL)
```

Don’t run the main analysis when interactive:

```python
if nbscript.argv is None:
    # Don’t do this stuff in Jupyter interface
```

Running with Slurm

Running as a script is great, but you need to submit to your cluster. nbscript comes with the command snotebook to make it easy to submit to Slurm clusters. It’s designed to work just like sbatch, but directly submit notebook files without needing a wrapper script.

snotebook is just like nbscript, but submits to slurm (via sbatch) using any Slurm options:

```
$ snotebook --mem=5G --time=1-12:00 input.ipynb
$ snotebook --mem=5G --time=1-12:00 input.ipynb argument1.csv
```

By default, this automatically saves to `input.out.TIMESTAMP.ipynb`, but can be configured.

You can put normal #SBATCH comments in the notebook file, just like you would when submitting with sbatch. But, it will only detect it from the very first cell that has any of these arguments, so don’t split them over multiple cells.

Example:

```
#SBATCH --mem=5G
#SBATCH --time=1-12:00
```

Just like with sbatch, you can combine command line options and in-notebook options.

See also

* nbscript main page, with more information.
8.1.5 Package your software well

This page gives hints on packaging your research software well, so that it can be installed by others.

As HPC cluster administrators, a lot of time is spent trying to install very difficult software. Many users want to use a tool released by someone, but it turns out to not be easy to install. Don’t let that happen to your code - keep the following things in mind, even at the beginning of your work. Do you want your code to be reused, so that you can be cited?

This page is specifically about packaging and distribution, and doesn’t repeat standard programming practices for scientists.

Watch a humorous, somewhat related talk “How to make package managers cry”.

Use the proper tools

Each language has some way(s) to distribute its code “properly”. Learn them and use them. Don’t invent your own way of doing things.

- Python (pip)
- Python (conda via conda-forge)
- R

Use the simplest, most boring, reliable, and mainstream system there is (that suits your needs).

Minimize dependencies

Build off of what others make, don’t re-invent everything yourself. But at the same time, see if you can avoid random unnecessary dependencies, especially ones that are not packaged well and well-maintained. It will make your life and others worse.

Don’t pin dependencies

Don’t pin exact versions of dependencies in a released library. Imagine if you want to install several different libraries that pin slightly different versions of their dependencies. They can’t be installed together, and the dependency solver may take a long time trying before it gives up.

But you do often want to pin dependencies for your environments, for example, the exact collection of software you are using to make your paper. This keeps your results reproducible, but is a different concept that releasing your software package.

You don’t pin dependencies strictly when someone may indirectly use your software in combination with arbitrary other packages. You should have some particular reason for each pin you have, not just “something may break in the future”. If the chances of something breaking in the future are really that high, you should wonder if you should recommend others to use this until that can be taken care of (for example, build on a more stable base).

You’ll notice that a lot of these topics deal with dependencies. Dependency hell is a real thing, and you should carefully think about them.
Be flexible on dependencies

Following up from above, be as flexible as dependencies as possible. Don’t expect the newest just because it’s the newest.

If you have to be strict on dependencies because the other software is changing behavior all the time, perhaps it’s not a good choice to build on. Maybe there’s no other choice, but that also means that you need to realize that your package isn’t as reusable as you might hope.

Try to be robust in dependencies

Follow the robustness principle to the extent possible: “Be conservative in what you do, be liberal in what you accept from others”. Try not to be as resistant as possible to dependencies changing, while providing a stable interface for other things. Of course, this is hard, and you need a useful balance. For “resistance to dependencies changing”, I interpret this as being careful what interfaces I use, and see if I can avoid using things I consider likely to change in the future.

Of course, robustness applies to other aspects, too.

Have tests

Have at least some basic automated tests to ensure that your code works in conjunction with all the dependencies. Perhaps also have a minimal example in the README file that someone can use to verify that they installed properly (could be the same as the tests). The tests don’t have to be fancy, even something that runs the code in a full expected use case will let you detect major problems early. This way, when someone is installing the software for someone else, they can check if they did it correctly.

Don’t expect the latest OS

Don’t design only for the latest and greatest operating system: then, many people who can’t upgrade right away won’t be able to use it easily. Or, they’ll have to go through extra effort to install newer runtimes on their older operating system.

For example, I usually try to make my software compatible with the latest stable operating systems from one year ago, and latest Python packages from two years ago. This has really reduced my stress in moving my code around, even if it does mean I have to wait to wait to use some new features.

Test on different dependency versions/OSs/etc

This starts to get a little bit harder, but it’s good to test with diverse operating systems or versions of your key dependencies. This probably isn’t worth it in the very early phases, but it is easier once you start using continuous integration / automated testing. Look into these once you get advanced enough.

Most clusters have different and older operating systems that you’d use on your desktop computer.
A container does not replace good packaging

“I only support using the Docker container” does not replace good packaging as described above. At the very least, it assumes that everyone can use Docker/singularity/container system of the year on the systems they need to run on. Second, what happens if they need to combine with other software?

A container is a good way to make compute easier and move it around, but make good packaging first, and use that packaging to install in the container.

Other

There is plenty more you should do, but it’s not specific to the topic of this page. For example,

- Have versions and releases
- Use a package repository suitable to your language and tool.
- Have good documentation
- Have a changelog
- etc...

See also

- Video “How to make package managers cry”.
- https://softdev4research.github.io/4OSS-lesson/

8.1.6 Python

Note For triton specific instructions see triton python page. For Aalto Linux workstation specific stuff, see Aalto python page.

Python is widely used high level programming language that is widely used in many branches of science.

Python distributions

<table>
<thead>
<tr>
<th>Simple programs with common packages, not switching between Pythons often</th>
<th>Python to use</th>
<th>How to install own packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anaconda 2/3</td>
<td>pip install --user</td>
<td></td>
</tr>
</tbody>
</table>

| Most of the use cases, but sometimes different versions of modules needed | Anaconda 2/3 | conda environment + conda |

| Special advanced cases. | Python from module system | virtualenv + pip install |

There are two main versions of python: 2 and 3. There are also different distributions: The “regular” CPython that is usually provided with the operating system, Anaconda (a package containing cpython + a lot of other scientific software all bundled together), PyPy (a just-in-time compiler, which can be much faster for some use cases).

- For general scientific/data science use, we suggest that you use Anaconda. It comes with the most common scientific software included, and is reasonably optimized.
• PyPy is still mainly for advanced use (it can be faster under certain cases, but does not work everywhere). It is available in a module.

Installing your own packages with “pip install” won’t work unless you have administrator access, since it tries to install globally for all users. Instead, you have these options:

• `pip install --user`: install a package in your home directory (`~/.local/lib/pythonN.N/`). This is quick and effective, but if you start using multiple versions of Python, you will start having problems and the only recommendation will be to delete all modules and reinstall.

• Virtual environments: these are self-contained python environment with all of its own modules, separate from any other. Thus, you can install any combination of modules you want, and this is most recommended.
  – Anaconda: use conda, see below
  – Normal Python: virtualenv + pip install, see below

### Installing own packages: Virtualenv, conda, and pip

You often need to install your own packages. Python has its own package manager system that can do this for you. There are three important related concepts:

• `pip`: the Python package installer. Installs Python packages globally, in a user’s directory (`--user`), or anywhere. Installs from the Python Package Index.

• virtualenv: Creates a directory that has all self-contained packages that is manageable by the user themself. When the virtualenv is activated, all the operating-system global packages are no longer used. Instead, you install only the packages you want. This is important if you need to install specific versions of software, and also provides isolation from the rest of the system (so that you work can be uninterrupted). It also allows different projects to have different versions of things installed. virtualenv isn’t magic, it could almost be seen as just manipulating PYTHONPATH, PATH, and the like. Docs: https://docs.python-guide.org/dev/virtualenvs/

• conda: Sort of a combination of package manager and virtual environment. However, it only installed packages into environments, and is not limited to Python packages. It can also install other libraries (c, fortran, etc) into the environment. This is extremely useful for scientific computing, and the reason it was created. Docs for envs: https://conda.io/projects/conda/en/latest/user-guide/concepts/environments.html.

So, to install packages, there is pip and conda. To make virtual environments, there is venv and conda.

Advanced users can see this rosetta stone for reference.

### Anaconda

**Anaconda** is a Python distribution by Continuum Analytics. It is nothing fancy, they just take a lot of useful scientific packages and put them all together, make sure they work, and do some sort of optimization. They also include all of the libraries needed. It is also all open source, and is packaged nicely so that it can easily be installed on any major OS. Thus, for basic use, it is a good base to start with. virtualenv does not work with Anaconda, use conda instead.
Conda environments

A conda environment lets you install all your own packages. For instructions how to create, activate and deactivate conda environments see http://conda.pydata.org/docs/using/envs.html.

A few notes about conda environments:

- Once you use a conda environment, everything goes into it. Don’t mix versions with, for example, local packages in your home dir. Eventually you’ll get dependency problems.

- Often the same goes for other python based modules. We have setup many modules that do use anaconda as a backend. So, if you know what you are doing this might work.

- The commands below will fail:
  - `conda create -n foo pip` # tries to use the global dir, use the `--user` flag instead
  - `conda create --prefix $WRKDIR/foo --clone root` # will fail as our anaconda module has additional packages (e.g. via pip) installed.

Basic pip usage

`pip install` by itself won’t work, because it tries to install globally. Instead, use this:

```
pip install --user
```

**Warning!** If you do this, then the module will be shared among all your projects. It is quite likely that eventually, you will get some incompatibilities between the Python you are using and the modules installed. In that case, you are on your own (simple recommendation is to remove all modules from `~/.local/lib/pythonN.N` and reinstall). If you get incompatible module errors, our first recommendation will be to remove everything installed this way and not do it anymore.

Python: virtualenv

Virtualenv is default-Python way of making environments, but does **not** work with Anaconda.

```
# Create environment
virtualenv DIR

# activate it (in each shell that uses it)
source DIR/bin/activate

# install more things (e.g. ipython, etc.)
pip install PACKAGE_NAME

# deactivate the virtualenv
deactivate
```
8.1.7 Linux shell crash course

Note: This is a kickstart for the Linux shell, to teach the minimum amount needed for any scientific computing course. For more, see the linux shell course or the references below.

This is basic B-level: no prerequisites.

Watch this in video format

There is a companion video on YouTube, if you would also like that format (and a slightly longer one with more detail).

If you are reading this case, you probably need to do some sort of scientific computing involving the Linux shell, or command line interface. You may wonder why we are still using a command line today, but the answer is somewhat simple: once you are doing scientific computing, you eventually need to script and automate something. The shell is the only method that gives you the power to do anything you may want.

These days, you don’t need to know as much about the shell as you used to, but you do need to know a few important commands because the command line works when nothing else does - and you can’t do scripting without it.

What’s a shell?

It’s the old-fashioned looking thing where you type commands with a keyboard and get output to the screen. It seems boring, but the real power is that you can script (program) commands to run automatically - which is the point of scientific computing.

You type a command, which may include arguments. Output gets shown to the screen. Spaces separate commands and arguments. Example: cp -i file1.txt file2.txt. cp is the command, -i is an option, file1.txt and file2.txt are arguments.

Files are represented by filenames, like file.txt. Directories are separated by /, for example mydir/file.txt is file.txt inside of mydir.

Exercise: Start a shell. On Linux or Mac, the “terminal” application does this.

Editing and viewing files

nano is an editor which allows you to edit files directly from the shell. This is a simple console editor which always gets the job done. Use Control-x (control and x at the same time), then y when requested and enter, to save and exit.

less is a pager (file viewer) which lets you view files without editing them. (q to quit, / to search, n / N to research forward and backwards, < for beginning of file, > for end of file)
Listing and moving files

`ls` lists the current directory. `ls -l` shows more information, and `ls -a` shows hidden files. The options can be combined, `ls -la` or `ls -l -a`. This pattern of options is standard for most commands.

`mv` will move or rename files. For example, `mv file.old file.new`.

`cp` will make a copy of a file, with the exact same syntax as `mv: cp file.old file.copy`.

`rm` will remove a file: `rm file.txt`. To remove a directory, use `rm -r`. Note that `rm` does not have backups and does not ask for confirmation!

`mkdir` makes a directory: `mkdir dirname`.

Current directory

Unlike with a graphical file browser, there is a concept of current working directory: each shell is in a current directory. If you `ls`, it lists files in your current directory. If a program tries to open a file, it opens it relative to that directory.

`cd` `dirname` will change working directories for your current shell. Normally, you will `cd` to a working directory, and use relative paths from there. `/` alone refers to the root directory, the parent of all files and directories.

`cd ..` will change to the parent directory (dir containing this dir). By the same token, `../..` the parent of the parent, and so on.

Exercise: Change to some directory and then another. What do `(cd -)` and `(cd with no arguments)` do? Try each a few times in a row.

Online manuals for any command

`man` is an on-line manual. type `man ls` to get help on the `ls` command. The same works for almost any program. Some are easy to read, some are impossible. In general you look for what you need, not read everything. Use q to quit or / to search (n and N to search again forward and backwards).

`--help` or `-h` is a standard argument that prints a short help directly.

Exercise: briefly look at the manual pages and `--help` output for the commands we have learned thus far. How can you make `rm` ask before removing a file?

History and tab completion

Annoyed at typing so much? We’ve got two ways to make work faster.

First, each shell keeps its (shell) history. By pushing the up arrow key, you can access previous lines. Never type similar things twice, go up in history and find the previous line, modify it, then push enter to re-run.

Shells also have tab completion. Type the first few letters of any command or filename and push tab once or twice… it will either complete it or show you the options. This is so important that it’s used often, and many command arguments can also be completed.

Exercise: Play around with tab completion. Type `pytho` and push TAB. (erase that then start over) Then type `p` and push TAB twice. (erase that and start over) Then `ls`, space, and the first few letters of a filename, then push TAB.
Variables

There are two kinds of variables in shell: **environment variables** and **shell variables**. You don’t need to worry about the difference now. The `$NAME` or `${NAME}` syntax is used to access the value of a variable.

For example, the environment variable `HOME` holds your home directory, for me `/home/rkdarst`. The command `echo` prints whatever its arguments are, so `echo $HOME` prints my home directory. (Note that the variable is a property of the `shell`, not of the `echo` command - this is sometimes important).

To set a variable, use `NAME=value`. `export NAME=value` sets it as an **environment variable** which means that other processes you start (from this shell) can use it.

The `$VARIABLE` syntax is also often used for examples: in this case, it isn’t an environment variable, but just something you need to substitute yourself when running a command.

See also

- The [linux shell course](#) has much more detail.
- Software Carpentry has a [basic shell course](#). Sections one to three are details of what is above (the rest is about shell scripting).

**Exercise**: for some fun, look at the manual pages for `cat`, `head`, `tail`, `grep`.

**Exercise (advanced)**: read the *Linux shell course* and understand what “pipes” and “piping” are.

### 8.1.8 SSH

This walk-through presumes that the user

- is working on a Linux machine or Mac
- has OpenSSH installed: `ssh -V` in the terminal to check
- has an account on the server of interest
- is connected to the Aalto network

We will be focusing on connecting to *Triton* but the methods described below are applicable to any of Aalto’s other remote servers.

**Basic use: connect to a server**

The standard login command is `ssh login_name@host_name`, where `login_name` is your standard Aalto login and `host_name` is the address of the server you wish to connect. In the case of Triton, the `host_name` is `triton.aalto.fi`. 
First time login

For Triton, you will be prompted to affirm that you wish to `ssh` into this server for the first time.

```plaintext
The authenticity of host 'triton.aalto.fi (130.233.229.116)' can't be established.
ECDSA key fingerprint is SHA256:04Wt813WFsYj27KiAyo3u6RiGBe1q1R19oJd2GXIAh0.
Are you sure you want to continue connecting (yes/no)?
```

Compare the key fingerprint you get to the one for the machine at this link, and if they do not match, please contact SciComp IT immediately. If they do match, type `yes` and press enter. You will receive a notice

```plaintext
Warning: Permanently added 'triton.aalto.fi,130.233.229.116' (ECDSA) to the list of
→ known hosts.
```

The public key that identifies Triton will be stored in `~/.ssh/known_hosts` and you ought not get this prompt again. You will be also asked to input your Aalto password before you are fully logged in.

Known servers

You will not receive an authenticity prompt upon first login if the server’s public key can be found in a list of known hosts. To check whether a server, Kosh for example, is known

```plaintext
ssh-keygen -f /etc/ssh/ssh_known_hosts -F kosh.aalto.fi
```

SSH keys: better than just passwords

By default, you will need to type your password each time you wish to `ssh` into Triton, which can be tiresome, particularly if you regularly have multiple sessions open simultaneously. A more secure (and faster) way to authenticate yourself is to use a ssh key pair and encrypt the this with a strong password. xkcd has good and amusing recommendations on the subject of passwords. This authentication method will allow you to log into multiple ssh sessions while only needing to enter your password once, saving you time and keystrokes.

Generate an SSH key

While there are many options for the key generation program `ssh-keygen`, here are the four main ones.

- `-t` -> the cryptosystem used to make the unique key-pair and encrypt it.
- `-b` -> the number of key bits
- `-f` -> filename of key
- `-C` -> comment on what the key is for

Here are our recommended input options for key generation.

```plaintext
ssh-keygen -t rsa -b 4096 -f ~/.ssh/id_rsa_triton -C "triton key for ${USER}"
```

After running this command in the terminal, you will be prompted to enter a password. PLEASE use a strong unique password. Upon confirming the password, you will be presented with the key fingerprint as both a SHA256 hex string as well as randomart image. Your new key pair should be found in the hidden `~/.ssh` directory. If you wish to use keys for other servers, you should generate new key pairs and use different passwords.
Copy public key to server

In order to use your key-pair to login to Triton, you first need to securely copy the desired public key to the machine with `ssh-copy-id`. The script will also add the key to the `~/.ssh/authorized_keys` file on the server. You will be prompted to enter your Aalto password to initiate the secure copy of the file to Triton.

```
ssh-copy-id -i ~/.ssh/id_rsa_triton.pub login_name@triton.aalto.fi
```

Login with SSH key

To avoid having to type the decryption password, the private key it needs to be added to the `ssh-agent` with the command

```
ssh-add ~/.ssh/id_rsa_triton
```

Once the password is added, you can ssh into Triton as normal but will immediately be connected without any further prompts. If you are unsure whether a `ssh-agent` process is running on your machine, `ps -C ssh-agent` will tell you if there is. To start a new agent, use `eval $(ssh-agent)`.

Config file: don’t type so many options

Remembering the full settings list for the server you are working on each time you log in can be tedious. A ssh config file allows you to store your preferred settings and map them to much simpler login commands. To create a new user-restricted config file

```
touch ~/.ssh/config && chmod 600 ~/.ssh/config
```

For a new configuration, you need specify in config at minimum the

- Host: the name of the settings list
- User: your login name when connecting to the server
- Hostname: the address of the server

So for the simple Triton example, it would be:

```
# Configuration file for simplifying SSH logins
#
# HPC slurm cluster
Host triton
  User LOGIN_NAME
  Hostname triton.aalto.fi
```

and you would use `ssh triton` to log in. Any additional server configs can follow the first one and must start with declaring the configuration Host:

```
# general login server
Host kosh
  User LOGIN_NAME
  Hostname kosh.aalto.fi
# light-computing server
Host brute
  User LOGIN_NAME
  Hostname brute.aalto.fi
```
There are optional ssh settings that may be useful for your work, such as:

```plaintext
# Turn on X11 forwarding for Xterm graphics access
ForwardX11 yes
# Connect through another server (eg Kosh) if not connected directly to Aalto network
ProxyJump LOGIN_NAME@kosh.aalto.fi
# Specify which ssh private key is used for login identification
IdentityFile ~/.ssh/id_rsa_triton
```

**Full sample config file**

This is placed in `~/.ssh/config`:

```plaintext
# general login server
Host kosh
    User LOGIN_NAME
    Hostname kosh.aalto.fi

# Triton, via kosh
Host triton
    User LOGIN_NAME
    Hostname triton.aalto.fi
    ProxyJump kosh
```

Now, you can just do `ssh triton` or `rsync triton:/m/cs/scratch/some_file .` directly, by using the `triton` alias. Note that the Triton rule uses the name `kosh` which is defined in the first part of the file.

**References**

- [https://www.ssh.com/ssh/](https://www.ssh.com/ssh/)
- [https://blog.0xbadc0de.be/archives/300](https://blog.0xbadc0de.be/archives/300)
- [https://www.mn.uio.no/geo/english/services/it/help/using-linux/ssh-tips-and-tricks.html](https://www.mn.uio.no/geo/english/services/it/help/using-linux/ssh-tips-and-tricks.html)
- [https://infosec.mozilla.org/guidelines/openssh](https://infosec.mozilla.org/guidelines/openssh)

**8.1.9 The Zen of Scientific computing**

Have you ever felt like all your work was built as a house of cards, ready to crash down at any time?

Have you ever felt that you are far too inefficient to survive?

No, you’re not alone. Yes, there is a better way.
Production code vs research code

Yes, many things about software development may not apply to you:

- **Production code:**
  - you sort of know what the target is
  - code is the main result
  - must be maintainable for the future

- **Research code:**
  - you don’t know what the target is
  - code is secondary

But research code *often becomes important in the future*, so not all can be an unmaintainable mess…

Research code pyramid

I know that *not all* research code will be perfect.

But if you don’t build on a good base, you will end up with misery.

Yes, you can’t do everything perfectly

Not everything you do will be perfect. But it has to be good enough to:

- be correct
- be changed without too much difficulty
- be run again once reviews come in
- ideally, not wasted once you do something new

Even as a scientist, you need to know the levels of maturity so that you can do the right thing *for your situation*.

It takes skill and practice to do this right. *But it is part of being a scientist.*

This talk’s outline:

- Describe different factors that influence code quality
- Describe what the maturity levels are and when you might need them
What aspects can you improve?

Below are many different aspects of scientific computing which you can improve. Some are good for everyone. Some you may not need yet. Different levels of maturity are presented for each topic, so that you can think about what is right for you.

Version control

Version control allows you to track changes and progress. For example, you can figure out what you just broke or when you introduced a bug. You can always go back to other versions.

Version control is essential to any type of collaboration.

- L0: no version control
- L1: local repo, just commit for yourself
- L2: shared repo, multiple collaborators push directly
- L3: shared repo, pull-request workflow

Resources:

- Github, CodeRefinery Gitlab, your institution’s equivalent, and many more.
- CodeRefinery lessons, git-intro and git-collaborative
- Software Carpentry Git-novice lesson

Modular code

Modularity is one of the basic prerequisites to be able to understand, maintain, and reuse things - and also hard to get right at the beginning.

Don’t worry too much, but always think about how to make things reusability.

- L0: bunch of copy-and-paste scripts
- L1: important code broken out into functions
- L2: separation between well-maintained libraries and daily working scripts.

Resources:

- CodeRefinery Modular Code Development lesson

Organized workspaces

You will need to store many files. Are they organized, so that you can find them later, or will you get lost in your own mess?

- L0: no particular organization system
- L1: different types of data separated (original data/code/scratch/outputs)
- L2: projects cleanly separated, named, and with a purpose

Resources:

- I don’t know of good sources for this.
• But you can find different recommendations for organizational systems

**Workflow/pipeline automation**

When you are doing serious work, you can’t afford to just manage stuff by hand. Task automation allows you to do more faster.

Something such as `make` can automatically detect changed input files and code and automatically generate the outputs.

• L0: bunch of scripts you have to run and check output of by hand.
• L1: hand-written management scripts, each output can be traced to its particular input and code.
• L2: `make` or other workflow management tool to automate things.
• L3: Full automation from original data to final figures and data

**Resources:**

• [CodeRefinery Reproducible Research lesson](https://www.coder refinery.org)

**Reproducibility of environment**

Is someone else able to (know and) install the libraries needed to run your code? Will a change in another package break your code?

Scientific software is notoriously bad at managing its dependencies.

• L0: no documentation
• L1: state the dependencies somewhere, tested to ensure they work
• L2: pin exact versions used to generate your results
• L3: containerized workflow or equivalent

**Resources:**

• [CodeRefinery Reproducible Research lesson](https://www.coder refinery.org)

**Documentation**

If you don’t say what you do, there’s no way to understand it. **You** won’t be able to understand it later, either.

At minimum, there should be some README files that explain the big picture. There are fancier systems, too.

• L0: nothing except scattered code comments
• L1: script-level comments and docstrings explaining overall logic
• L2: simple README files explaining big picture and main points (example)
• L3: dedicated documentation including tutorials, reference, etc.

**Resources:**

• [CodeRefinery Documentation lesson](https://www.coder refinery.org)
Testing

You have to test your code at least once when you first run it. How do you know you don’t break something later?

Testing gives you a way to ensure things always work (and are correct) in the future by letting you run every test automatically.

There’s nothing more liberating than knowing “tests still pass, I didn’t break anything”. It’s extremely useful for debugging, too.

- L0: ad-hoc and manually
- L1: defensive programming (assertions), possibly some test data and scripts
- L2: structured, comprehensive unit/integration/system tests (e.g. pytest)
- L3: continuous integration testing on all commits (e.g. Github Actions)

If code is easy to test, it is usually easy to reuse, too. Furthermore, making code testable makes it reusable.

Resources:
- CodeRefinery Testing lesson:
- GitHub Actions

Licensing

You presumably want people to use your work so they will cite you. If you don’t have a license, they won’t (or they might and not tell anyone).

Equally, you want to use other people’s work. You need to check their licenses.

- L0: no license given / copy and paste from other sources
- L1: license file in repo / careful to not copy incompatible code
- L2: license tracked per-file and all contributors known.

Resources:
- CodeRefinery Software social coding
- https://choosealicense.com/

Distribution

Code can be easy to reuse, but not easy to get. Luckily there are good systems for sharing code.

- L0: code not distributed
- L1: code provided only if someone asks
- L2: code on a website
- L3: version control system repo is public
- L4: packaged, tagged, and versioned releases

Resources:
- Python: Packaging tutorial
- Similar for any other language you may use
Reuse

Are you aware of what others have already figured out through their great effort?
Choosing the right thing to build off of is not always easy, but you must
  • L0: reinvent everything yourself
  • L1: use some existing tools and libraries
  • L2: deep study of existing solutions and tools, reuse them when appropriate

Resources:
  • I don’t know where to refer you to right now.

Collaboration

Is science like monks working in their cells, or a community effort?
These skills move so fast that learning peer-to-peer is one of the best ways to do it.
There’s a whole other art of applying these skills which isn’t taught in classes.
If you don’t work together, you will fall behind.
  • L0: you work alone and re-invent everything
  • L1: you occasionally talk about results or problems
  • L2: collaborative package development
  • L3: code reviews, pair programming, etc.
  • L4: community project welcoming other contributors

Resources:
  • Most every CodeRefinery lesson
  • Plenty more

The future

Science with computers can be extremely enjoyable… or miserable.
We are here to help you. You are here to others.
Will we?

8.1.10 Practical git PRs for small teams

This is the prototype of a mini-course about using git for pull requests (PRs) within small teams that are mostly
decentralized, perhaps don’t have test environments everywhere, and thus standard review and CI practices don’t
directly apply. The audience is expected to be pretty good with git already, but wondering how PRs apply to them.
The goal isn’t to convince you to use PR-based workflows no matter the cost, but instead think about how the tech can
make your social processes better.
Status: Alpha-quality, this is more a start of a discussion than a lesson. Editor: rkdarst
Learning objectives

• Why use pull requests?
• What are the typical procedures of using PRs?
• How do we adapt our team to use them?
• How does this improve our work?

Why pull requests?

pull request = change proposal

You have some work which should be reviewed before deploying.

• Someone is expected to give useful feedback
• Maybe a quick idea, easier to draft&discuss than talk about it abstractly

pull request = review request

You’ve made the change already, or you are already the expert so don’t expect it to really be debated.

• You edited it in deployment, or it is already live
• Or you are the expert, and others don’t usually give suggestions
• Still, someone might have some comments to improve your integration with other services.

pull request = change announcement

• You don’t expect others to ever make suggestions
• But you think others should know what you are doing, to distribute knowledge
• If no one comments, you might merge this yourself in a few hours or days.

pull request = CI check

• You want the automated tests/ continuous integration (CI) to run to verify the change works.
• If it works, you might merge yourself even without others knowing.
• A bit safer than CI after the push to master.
Benefits of PRs

• Multiple sets of eyes
  – Everything should be seen by multiple people to remove single point of failure problems.
  – Share knowledge about how our services work.
  – Encourages writing a natural-language description of what you are doing - clarify purpose to yourself and others

• Suggestion or draft
  – Unsure if good idea, make a draft to get feedback
  – Discuss and iterate via issue. No pressure to make it perfect the first time, so writing is faster

• CI
  – Run automated tests before merging
  – Requires a test environment
  – Very important for fast and high-quality development.

• Discussion
  – Structured place for conversation about changes
  – Refer to and automatically close issues

How do you make a pull request

• Technically, a pull request is:
  – A git branch
  – Github/Gitlab representation of wanting to merge that head branch into some base branch (probably the default branch).
  – Discussion, commenting, and access control around that
  – So, there’s nothing really magic beyond the git branch.

• We don’t really need to repeat existing docs: you can read how to on Github, Gitlab, etc. yourself.

• A PR starts with a branch pushed to the remote.

• Then, the platform registers a pull request which means “I want to merge this branch into master”. (Yes, a bit misnamed) Go to the repo page and you see a button, or a link to make one is printed when you push.

• git-pr makes it easy - fewest possible keystrokes, no web browser needed, and I use the commit message also as the PR message to save even more time.
Pull request description

- These days, I (rkdarst) tend to write my initial PR message into my commit, then `git-pr` will use that when I push. This also stores the description permanently in the git history.
- There is also the concept of “pull request templates” within Github/Gitlab. (They can keep changes organized, provide checklists, and keep things moving. But after fast small PRs via `git-pr` I really don’t like this being required for small changes where I can write the important aspects myself.)
- What should go in a description:
  - Why are changes being made?
  - What are the changes?
  - Risks, benefits, etc…
  - Is it done or a work in progress? Need help?
  - What should be reviewed?

CI checks

- CI pipelines can run on the pull request and will report failures. On Github, success is a green check. Can be shared with checks of direct pushes.
- Even if there aren’t tests, syntax checks and similar could be useful.

Semantics around PRs

How do you actually review and handle a PR once it comes in? What’s the social process?

Actions you can take

Actions you can do from the web (Github):

- **merge**: accept it
- **comment**: add a message
- **approve/request changes**: “review” you can do from “file list” view
- **line comments**: (*): from diff view, you can select ranges of lines and comment there
- **suggestions**: (*): from diff, you can select ranges of lines then click “suggest” button to make a suggestion. This can easily be applied from web.
- **commit suggestion**: (*): from diff view, you can accept the suggestion and it makes a commit out of it.
- (*): items can be done in batch from file view, to avoid one email for every action.
- **draft** pull request can’t be merged yet. There is a Github flag for this, or sometimes people prefix with WIP:.
- **assign a reviewer**: request people to do the review, instead of waiting for someone to decide themselves.
- **close**: Reject the change and mark the PR as closed.
My usual procedure

- If it’s good as-is, just click “merge”
  - If it’s a new contributor I usually try to say some positive words, but in long-term efficient mode, I don’t see a need to.
- Otherwise, comment in more detail. Line-based comments are really useful here. Commenting can be line-based, or an overall “accept”, “request changes”, or “comment” on the PR as a whole (see above)
- If you aren’t sure if you are supposed to merge it (yet), but it looks good, just “approve” it.
  - This can be a sign to the original author that it looks sane to you, and they merge when they are ready.
- If someone marks my PR “approve” but don’t merge it themselves, I will merge it myself as soon as I am ready.
- If someone else requested changes, I’ve done the changes (if I agree), and I think there’s not much more to discuss, I will just merge it myself without another round of review.
- You can both make suggestions and approve (usually with some words saying no need to accept the suggestions if they don’t make sense).

How do humans use PRs?

Who should merge them?

- What happens when the person making the PR is the only one (or main one) who can give it a useful review?
  - Then, perhaps your team needs some redundancy…
- You can assign reviewers, if you want to suggest who should take a look.
- Discuss as part of your team for each project. This leads to a social discussion of “how do we collaborate in practice?”

When do you merge a pull request?

- How much review do you need to give, if you aren’t the expert?
- My proposal:
  - If you are aren’t the author, and can evaluate it, merge it ASAP
  - If you aren’t an expert, but no one else has merged it after a few days, merge it yourself. Or if you are the original author and need it.
  - If no one else has after a week, anyone does it (mainly relevant to external contributors).
- I don’t feel bad making a PR if I expect I will be the one to merge it a few days later: at least I gave people a chance to take part.
How do you keep up to date with PRs?

- this view lists open Github PRs in an organization

How can our team adapt to PRs?

Traditional software project or utility

- PRs make a lot of sense

Deployments: There is no testing environment!

Yes, there should be a test environment, but let’s be real: many thing start off too small to have that. What do we do about it?

- “If the change has already been made, it’s not really a change proposal”
- PRs don’t work too well here, but when you think about it, it would be nice to be able to test before deploying!
  - Maybe this gives us encouragement to use more PRs
- Make a PR anyway even though it’s in productive, as a second-eyes formality.

All of our projects are independent

- Is this good for knowledge transfer?

What advantages would we see with more PRs?

Other

These things can make our work a bit soother, and something we can discuss.

**git-pr**

- I got annoyed at needing too many keystrokes, and having to go to a web browser to create the pull requests
- I created git-pr to make this as fast as possible, and it really does feel much smoother now
- Works equally for Github and Gitlab, at least.

**Shared git aliases**

- How can we deploy some shared aliases to all hosts we manage, to make git more enjoyable to use?
Blocking authorless commits

- To block authorless commits, run this to set a pre-commit hook:

```bash
echo 'git var GIT_AUTHOR_IDENT | grep root && echo "Can not commit as root! Use --author" && exit 1 || exit 0' >> .git/hooks/pre-commit ; chmod a+x .git/hooks/pre-commit
```

- Can this be made automatic in all of our repos?

Cheatsheets: git for normal people, Gitlab (produced by Gitlab, with Aalto link)
We have various recommended training courses for researchers who deal with computation and data. These courses are selected by researchers, for researchers and grouped by level of skill needed.

9.1 Training

Scientific computing and data science require special, practical skills in programming and computer use. However, these aren’t often learned in academic courses. This page is your portal for getting these skills. The focus is practical, hands-on courses for scientists, not theoretical academic courses.

9.1.1 Scientific Computing in Practice

SCIP is a lecture series at Aalto University which covers hands-on, practical scientific computing related topics. Lectures are open for the entire Aalto community as well as our partners at FGCI consortium.

Examples of topics covered at different lectures: HPC crash course, Triton kickstarts, Linux Shell, Parallel programming models: MPI and OpenMP, GPU computing, Python for scientists, Data analysis with R and/or Python, Matlab, HTCondor and many others.

March 2021 / MPI introduction

Part of Scientific Computing in Practice lecture series at Aalto University.

Audience: Employees and students looking for the extensive intro into MPI programming model.

About the course: In this introduction to MPI programming we are going to cover all the basic functionalities of the API, and see how these are applied to real problems. Starting from simple, well-established algorithms, we will build up towards more advanced and realistic examples commonly found in computational physics and chemistry. Finally, we will explore how the strengths of OpenMP multithreading and MPI can be combined with hybrid parallel programming.

The course is focused on practical aspects of parallel programming applied to problem solving. The language of choice is C, but few words will be spent on how to use MPI in FORTRAN if required by the students. Students are expected to have a basic understanding of conventional programming, and some familiarity with C.

Lecturer: Filippo Federici, D. Sc., Department of Applied Physics, Aalto University

Time, date:
Aalto scientific computing guide

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tue 9.3</td>
<td>10:00-13:00</td>
</tr>
<tr>
<td>Fri 12.3</td>
<td>10:00-13:00</td>
</tr>
<tr>
<td>Tue 16.3</td>
<td>10:00-13:00</td>
</tr>
<tr>
<td>Fri 19.3</td>
<td>10:00-13:00</td>
</tr>
</tbody>
</table>

Place: Zoom link provided to registered participants

Cost: Free of charge for FGCI consortium members and Aalto employees, students.

Registration: Click here to register

Credits (opintopisteet): Credits available for the Aalto students and course certificate can be provided on request for the outsiders. Since the main focus of the course is learning through experience, the students will be required to perform simulations for homework and hand in result report. Credits/certificate require full time participation and handling assignments. Full course hours correspond roughly to 1 ECTS.

Other comments: Participants from Aalto university will be provided with access to Triton for running examples. We will use SSH for Triton connection. One can follow the theoretical part without access to the Triton cluster, but we highly recommend doing the tutorials together. If you cannot access the Triton HPC cluster, you need to install in your system an MPI library and a reasonable C compiler (GCC>5.4). All the functions we use should be available in any MPI implementation.

Additional course info at: filippo.federici -at- aalto.fi / ivan.degyarenko -at- aalto.fi

March 2021 / Linux Shell Scripting

Part of Scientific Computing in Practice lecture series at Aalto University.

Audience: employees and students, intermediate or advanced level in Linux/Mac shell. (We happily accept public attendance, but are doing limited advertisement right now - if the course was advertised to you, welcome.)

About the course: Did you know that SLURM batch script is a regular BASH shell script mostly? Did you know that when you login to Triton (our computer cluster) your terminal is a fully enabled shell scripting environment? Do you want to know how to create an alias, a function, how to use loops and traps in a shell, work with variables and arrays? This course is oriented on those who want to start using BASH programming fully and use terminal efficiently.

We expect that course participants are familiar with the shell basics (experience with BASH, ZSH, etc). We somewhat touch the Part 1 of the Linux Shell tutorial, and continue to Part 2. Though we expect that participant knows how to create a directory and can edit file from the linux shell command line. We will be scripting a lot, there will be lots of demos and real practicing.

Lecturer: Ivan Degtyarenko, D. Sc., Science IT / Department of Applied Physics, Aalto University

Place: Online, common Zoom link for all the sessions aalto.zoom.us/j/947220751

Time, date (all times EET):

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon 22.03</td>
<td>10:00-13:00</td>
</tr>
<tr>
<td>Wed 24.03</td>
<td>10:00-13:00</td>
</tr>
<tr>
<td>Mon 29.03</td>
<td>10:00-13:00</td>
</tr>
<tr>
<td>Wed 31.03</td>
<td>10:00-13:00</td>
</tr>
</tbody>
</table>

Course material: will be mostly based on the second part of the scicomp.aalto.fi/training/linux-shell-tutorial.

Additional materials and homework assignments at triton.aalto.fi:/scratch/scip/BASH (will be made available for others)
**Cost:** Free of charge for FGCI consortium members including Aalto employees and students.

**Registration:** You can register at this link

**Credits:** Credits available for the Aalto students and course certificate can be provided on request for the outsiders. Credits/certificate require full time participation and handling homework/assignments. Full course hours correspond roughly to 1 ECTS.

**Setup instructions:** For the online course we expect you to have Zoom client installed on your local workstation/laptop. Then we expect you to have access to Linux-like shell terminal. You can check BASH installation instructions for various operating systems at this link. If needed participants can be provided with access to the Triton HPC cluster for running examples.

**Additional course info at:** ivan.degtyarenko -at- aalto.fi

---

**April 2021 / Hands-on Data Anonymization**

Part of *Scientific Computing in Practice* lecture series at Aalto University and Aalto Data Agents RDM training.

**Audience:** Anyone who works with personal data in all its forms (background variables from questionnaires, medical images, health data, geospatial location data, speech, videos, pictures, etc.).

**About the course:** The goals for this course are practical: to have people to actually de-identify/pseudo-anonymise/anonymize personal data in many of its forms and also use modern techniques for working with personal/sensitive data. There will be a conceptual introduction on day1, other days will cover tools for (pseudo)anonymizing personal data.

**Lecturers:** Enrico Glerean, Dan Häggman, Abraham Zewoudie

**Time, date, place:** the course consists of four online hands-on sessions. Zoom link to be posted to the registered participants list

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.04</td>
<td>12:00-15:00</td>
<td>Basics of Anonymization and working with participants background data</td>
</tr>
<tr>
<td>15.04</td>
<td>12:00-15:00</td>
<td>Automating anonymisation for tabular data</td>
</tr>
<tr>
<td>19.04</td>
<td>12:00-15:00</td>
<td>Anonymization for complex datasets: Faces in pictures and videos, Speech, Geospatial data</td>
</tr>
<tr>
<td>22.04</td>
<td>12:00-15:00</td>
<td>Anonymization for medical data; advanced techniques for working with sensitive data (data synthesis, federated learning and differential privacy)</td>
</tr>
</tbody>
</table>

**Course material:** LINK TBA

**Cost:** Free of charge

**Registration:** https://forms.gle/JbC4NB6aPazYtZ7G8

**Credits:** Course certificate and 1 ECTS for those who have a student ID.

**Additional course info at:** enrico.glerean -at- aalto.fi
April 2021 / Software design for scientific computing

Part of *Scientific Computing in Practice* lecture series at Aalto University.

**Audience:** employees and students, intermediate or advanced level in Python. For this course we also warmly invite those who already know everything there is to know about Python.

**About the course:** Getting the desired end result is an important first step in writing your analysis script or program. But it is just the beginning of the journey to truly great software. In this course we set you on a path to thinking about the *design* of your code: how to make it obvious what the code does, that it is correct, efficient and elegant. As programmers, we are on this journey for our entire career. For example, we assume you know *how* to write a function in Python. In this course, we aim to teach you *which* function you should write.

We will present some design guidelines and discuss them together. Then, we will all implement a simple, but not trivial, data analysis pipeline from the neuroimaging domain. Next, we will review each-other's code based on the design guidelines and note things that were designed well and things that could be improved. Finally, we will re-work our code based on the feedback we received and things we learned from reading other people’s code. Hopefully, you will end up with a pearl of great code that can serve as inspiration for the code you’ll write from here on after.

We expect that course participants are familiar with the Python programming language, along with the basic packages for scientific computing (NumPy/SciPy/Matplotlib/Pandas). To test your knowledge of these basics (and point you to relevant documentation to fill in any gaps), we have designed the Gizmo challenge.

**Lecturers:**
- Susanne Merz, NBE, Aalto University
- Marijn van Vliet, Science IT, Aalto University

**Place:** Online, common Zoom link for all the sessions (Zoom link will be sent after registration).

**Time, date (all times EET):**

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.04</td>
<td>12:00-14:00</td>
<td>Theory session</td>
</tr>
<tr>
<td>26.04</td>
<td>10:00-14:00</td>
<td>First review sessions (half hour slot per person)</td>
</tr>
<tr>
<td>03.05</td>
<td>10:00-14:00</td>
<td>Second round of review sessions (half hour slots)</td>
</tr>
<tr>
<td>03.05</td>
<td>14:00-15:00</td>
<td>Recap session and closing</td>
</tr>
</tbody>
</table>

**Course material:** All course material can be found in this repository: https://github.com/susamerz/CDWAssignment.

**Cost:** Free of charge for FGCI consortium members including Aalto employees and students.

**Registration:** currently closed.

**Credits:** Credits are available for Aalto students and a course certificate can be provided on request for outsiders. Credits/certificate require full time participation and handling home work/assignments. Full course hours correspond roughly to 1 ECTS.

**Setup instructions:** To access the online course you need to have access to Zoom, either through the Zoom client or through a browser. To follow and participate in the workshop, we expect you to also have access to a Python installation with the basic scientific software stack (NumPy/SciPy/Matplotlib/Pandas, see https://www.scipy.org). We recommend an anaconda installation. You can refer to https://coderefinery.github.io/installation/python/ for installation instructions, ignoring the CodeRefinery specific parts. You will also need a working and configured git installation. Instructions at https://coderefinery.github.io/installation/git/.

**Additional course info at:** susanne.merz -at- aalto.fi or marijn.vanvliet -at- aalto.fi
May 2021 / Matlab Advanced

Part of Scientific Computing in Practice lecture series at Aalto University.

Audience: Everyone with basic knowledge of Matlab

About the course:

MATLAB is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis and numerical computation in general. — The Mathworks Inc.

Teaching will be interactive, “learning by doing”.

Course webpage

Matlab Advanced 2021 webpage

Schedule:

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tue 11.05</td>
<td>12:00-15:00</td>
<td>Loading and handling very large datasets</td>
</tr>
<tr>
<td>Wed 12.05</td>
<td>12:00-15:00</td>
<td>Advanced programming techniques</td>
</tr>
<tr>
<td>Tue 18.05</td>
<td>12:00-15:00</td>
<td>Parallelisation on CPUs/GPUs/HPC</td>
</tr>
<tr>
<td>Wed 19.05</td>
<td>12:00-15:00</td>
<td>Making high quality figures</td>
</tr>
</tbody>
</table>

Place: Online, zoom link will be sent to registered participants

Lecturers: Enrico Glerean, D.Sc., Aalto Scientific Computing

Registration:

Click here for registering to Matlab Advanced 2021

Cost: Free of charge for FGCI consortium members including Aalto employees and students.

No-show: If you registered and cannot come to the course anymore, please inform enrico.glerean@aalto.fi!

Course prerequisite requirements and other details:

The course is done online, you should use your laptop, Matlab is available for Aalto staff and students at download.aalto.fi (Linux/Mac/Windows versions).

Course credits: if you have an Aalto student number, you can obtain 1 ECTS credit for the course (12h attendance + homeworks). Check the Matlab Advanced course webpage for a detailed list of requirements for the credit.

Additional course info at: enrico.glerean -at- aalto.fi

June 2021 / Intro to Scientific Computing (FGCI HPC Summer Kickstart)

News

- Watch at https://twitch.tv/coderefinery
- The livestream is archived on Twitch for 14 days. Videos will be posted on this playlist once they are ready.

Before the workshop:

- Registration is open: https://forms.gle/yNFLYt676kKorF3X7
- View the prerequisites below.
Check back here for other updates that don’t get their own email.

Part of the *Scientific Computing in Practice* lecture series at Aalto University.

**Audience:** All researchers looking for a start to scientific computing. We go over the various options and tools that everyone needs to know about, and then go in-depth about using a remote computational cluster (though these skills will be useful to everyone). This is specifically designed for our summer workers who are just starting their internship, but anyone who is doing computing or data-focused work can get something from this course. Anyone is welcome to listen along and learn from some experts.

Most examples use Aalto University resources, but everyone can learn something and we are careful to explain local vs general practices.

**About the course:**

Summer Kickstart is a three day courses for researchers to get started with the available computational resources at FGCI (Finnish Grid and Cloud Infrastructure, basically HPC, high-performance computing, at universities) and CSC (the Finnish national computing center). On the day one we start with the basic HPC intro, and some basic intro to Linux command line and Git version control, for those who are not yet familiar with these tools.

On days two and three we cover one by one steps on how to get started on the local computational clusters: learning by doing with lots of examples and hands-on exercises.

By the end of the course you get the hints, ready solutions and copy/paste examples on how to find, run and monitor your applications, and manage your data. In addition to how to optimize your workflow in terms of filesystem traffic, memory usage etc.

University specific information:

- Aalto: this course is obligatory for all new Triton users and recommended to all interested in scientific computing in general. *Basic reference information is at the Triton page*
- University of Helsinki
- Tampere: this course is recommended for all new Narvi users and also all interested in HPC. Most things should work with simply replacing triton -> narvi. Some differences in configuration are listed in *Narvi differences*

**Practical information**

**Time, date:** Mon 7.6, Tue 8.6, Wed 9.6, 11:50-16:00 EEST

**Place:** Online, see below

**Lecturer by:** Aalto Scientific Computing (Science-IT) and others

**Registration:** [https://forms.gle/yNFLYt676kKorF3X7](https://forms.gle/yNFLYt676kKorF3X7)

**Cost:** Free of charge for FGCI consortium members including Aalto employees and students. Livestream is free to everyone.

**Additional course info at:** scip@aalto.fi
How to attend

This is an online hybrid of MOOC and interactive:

- **Livestream**: anyone may watch at https://twitch.tv/coderefinery, no registration needed!
- **Zoom**: if you register, you will be able to attend a Zoom meeting that includes interactive breakout rooms and hands-on help. We watch the livestream for the main material.
- **HackMD**: instead of chat, this is used for Q&A. See the CodeRefinery HackMD manual for how this works.

Schedule

The daily schedule will be adjusted based on the audience; below is the tentative plan. There will be frequent breaks. You will be given time to try and ask, it’s more like an informal help session to get you started with the computing resources. All times are EEST (Helsinki) time.

- **Preparatory material**: Each year the first day has varying topics presented. We don’t repeat these every year, but we strongly recommend that you watch these videos yourself as preparation:
  - Basic Linux shell and scripting
  - Scientific computing workflows

- **Day #1 (Mon 7.jun)**: Basics and background
  - 11:50: **Joining time and pre-discussion**, please join 10 minutes early. (Richard Darst, Enrico Glerean)
  - 12:00: **Welcome, general introduction** (Notes) (Enrico Glerean and all)
  - 12:10: **HPC crash course: what is behind the front-end** HPC fundamentals: terminology, architectures, interconnects, infrastructure behind, as well as MPI vs shared memory. Continued on day 3. (Ivan Degtyarenko, Simppa Akaslompolo) (Slides (.pdf))
  - 12:40: **Summary and discussion about the videos “Basic linux shell scripting” and “Scientific computing workflows”** (see videos in preparatory material above) (Notes) (Richard Darst, Enrico Glerean)
  - 12:50: Break
  - 13:00: **Currently available resources at CSC** CSC is the Finnish center for scientific computing, and also has many resources for research. (Slides) (Jussi Enkovaara, CSC).
  - 13:45: Break
  - 14:00: **Git intro**: why you need version control for any scientific work and how to get started. We don’t go in depth into theory, but talk about the simplest usage by yourself. (Richard Darst, Jarno Rantaharju)
  - 14:45: Break
  - 15:00: **Your future career in scientific computing (and this course)**. (Notes) (Enrico Glerean, TBA)
  - 15:15: **Connecting to the cluster**, hands-on. Get connected in preparation for day 2 (Enrico Glerean)
    - Aalto: *Connecting to Triton tutorial* – if you can ssh to Triton and run `hostname`, you are ready for tomorrow.
    - Helsinki: general information
    - Tampere: Connecting to Narvi

- **Day #2 (Tue 8.jun)**: Basic use of a cluster (Richard Darst, Simo Tuomisto)
  This day will go over all practical aspects of using the cluster
  - 11:50: Joining time/icebreaker
– 12:00: *Connecting to Triton*
  
  * Every site will have its own ways of connecting. The basic lessons of `ssh` is the same for everyone, but it will have a different hostname and possibly different initial steps (jump hosts).
  * Aalto: (same)
  * Helsinki: general information
  * Tampere: Connecting to Narvi. Note, that you will need SSH keys.

– 12:30: *Applications*
  
  * Each site will be quite different here, so don’t worry about making the exercises work outside of Aalto, but think and prepare for what comes next (where we’ll explain the differences).

– 12:50: Break

– 13:00: *Software modules*

– 13:20: *Data storage*
  
  * Aalto: (same)
  * Helsinki: general information
  * Tampere: Narvi storage
  
  * This topic is very site-specific. The general principles will apply everywhere, but the exact paths/servers will vary.

– 13:50: Break

– 14:00: Short talk: Radovan Bast (UiT The Arctic University of Norway): Asking for help with supercomputers
  
  * How should you write support requests so that you get quick (and useful!) answers? Radovan, one of the founders of CodeRefinery, will talk about how we can all improve the dialogue between supercomputer user community and support staff so that we always remain respectful and try to learn and solve problems together.

– 14:35: *Interactive jobs*
  
  * The basic Slurm concepts are the same across all clusters (at least all those that use Slurm, but that is everyone in Finland). However, partition names may be different. You can list partitions at your site using `sinfo -O partition` and list nodes at your site with `sinfo -N`. How these work will vary depending on your site - definitely read up on this.

– 14:50: Break

– 15:00 Continuing with interactive slurm jobs and exercises

– 16:00: End

• Day #3 (Wed 9.jun): Advanced cluster use (Simo Tuomisto, Richard Darst)
  
  – 11:50: Joining time/icebreaker
  
  – 12:00 *Serial Jobs*
  
  – Monitoring job progress and job efficiency
  
  – Array jobs
Array jobs allow you to quickly run many jobs, and are the simplest unit of advanced computing. We will go over them in detail.

- **Software modules**
  - In other sites, you should `module load fgci-common` to be able to make the Aalto modules available. Other specifics, such as `matlab`, won’t directly work.

- **GPU computing** (Simo Tuomisto)
  - Aalto: (same as above)
  - Helsinki: general information
  - Tampere: Narvi GPU computing differences
  - At other sites, you may need to use `-p gpu` in addition to `--gres=gpu`.

- **Parallel computing** (Simo Tuomisto)
- **Parallel computing programming** (Ivan Degtyarenko, Simo Tuomisto)

- 16:00: End

**Follow-up suggestions:** While not an official part of this course, we suggest these videos (co-produced by our staff) as a follow-up perspective:

- Attend a CodeRefinery workshop, which teaches more useful tools for scientific software development.
- Look at Hands-on Scientific Computing for an online course to either browse or take for credits.
- Cluster Etiquette (in Research Software Hour): The Summer Kickstart teaches what you can do from this course, but what should you do to be a good user.
- How to tame the cluster (in Research Software Hour). This mostly repeats the contents of this course, with a bit more discussion, and working one example from start to parallel.

**Prerequisites**

Participants will be provided with either access to their university’s cluster or Triton for running examples.

- You should have an account on your university’s HPC cluster:
  - Aalto: if you do not yet have access to Triton, request an account in advance.
  - Helsinki: Account notes at the bottom of this page
  - Tampere: your cluster will require ssh keys to connect.
  - Others: Aalto will provide you with a guest Triton account, check back for more information.

- Participants are expected to have a SSH client installed (for options, see the Triton connecting tutorial for examples).
- You should install Zoom. Hints on installation.
- If you aren’t familiar with the Linux shell, read the crash course, watch the video, or watch the relevant preparatory video linked as part of the schedule.
- Try to get connected to your cluster in advance. We have some time scheduled for this, but you need to also try in advance, or else we can’t keep up.
  - Aalto: connecting to Triton
  - Helsinki: general information
Other preparation

How to attend this course:

• Take this seriously. There is a lot of material and hands-on exercises. Don’t overbook your time, don’t skip hands-on parts, and come prepared.

• Anyone may watch via Livestream, https://twitch.tv/coderefinery. Register anyway to get emails.

• You will be given a Zoom link to join. Join each session 10 minutes early.

• Join with a name of “(University) First Last”, e.g. “(Aalto) Richard Darst”. This will help us to put people into university-specific breakout rooms.

• There will be a <HackMD.io> document sent to all participants. This is for communication and asking questions. Read more about how this works here
  – Always write new questions or comments at the bottom of the document.
  – Moderators will follow the developments, and answer questions and comments. You may get several answers from different perspectives, even. Our focus is the bottom, but we will scan the whole document and keep it organized.
  – The final document (excluding personal data and questions about individual circumstances) will be published as the notes at the end.

August 2021 / Introduction to Julia

Course basics

An introduction to the Julia programming language by Science IT at Aalto University.

The course takes place over two days, August 10 and 11 from 9:00 to 13:00. On the first day we introduce the Julia language itself and on the second day we go in some more detail into performance, types and the Julia package ecosystem.

Materials: Links and instructions TBA

Place: Online, zoom link will be sent to registered participants

Lecturers: Jarno Rantaharju, Research Software Engineer, Aalto Scientific Computing

Registration:

Please register at this link

Based on

• “Introduction to Julia” by CSC
SCIP Archive

Currently active (upcoming) courses have been moved to the training index. Below is a list of past courses.

2020

- MPI introduction (February 2020)
- Hands-on Molecular Dynamics with LAMMPS (February 2020)
- Linux shell scripting (March 2020)
- Matlab advanced (April 2020)
- Mega CodeRefinery (June 2020, materials, videos)
- FGCI kickstart (June 2020)
- Linux shell basics (September 2020)
- Python for scientific computing (September 2020, materials)
- Data analysis workflows with R and Python (October 2020, materials)
- CodeRefinery online (October 2020, materials)
- Matlab basics (November 2020, materials, videos)
- GPU computing in practice (December 2020)

2021

- Introduction to Data Analysis strategies at Aalto, Linux shell, HPC kickstart 2021 (Jan/Feb 2021, materials part 1, materials part 2, videos)

This course list is used to be at science-it.aalto.fi/scip page, but that page is now deleted. This series has existed since 2016.

Announcement maillist

Events and other Aalto Scientific Computing (Science-IT) announcements distributed over several lists such as the Triton-users and department mailing lists. In addition we run the scicomp-announcements@list.aalto.fi maillist that covers everyone else who wants to stay tuned and receive Science IT news.

The moderated list is free to subscribe / unsubscribe at any time, accepts all emails including non-Aalto ones.

- scicomp-announcements web page for users

Future courses Fall / Winter 2021-22 courses (tentative plan) - Linux Shell Basics, Python for Scientists, Data analysis with R and Python, Matlab Basics, GPU computing, Triton winter kickstart.

Anyone can sign up for announcements at the SCIP announcement mailinglist.
9.1.2 Other interesting courses

Data management, Reproducibility, open science

Please check the spring calendar for training on data management and open science, organised by the Aalto Data Agents:

- How to make your research/code reproducible | 2.3.2021 at 2-3.30 PM
- Introduction to research data management | 11.3.2021 at 10-11 AM
- How to store research data | 18.3.2021 at 10-11 AM
- Handling of personal data | 25.3.2021 at 10-11.30 AM
- Working with restricted datasets | 15.4.2021 at 10-11
- Responsible conduct of research, questionable research practices... and possible cures | 28.4.2021 at 10-11 AM
- How to share research data through a repository: A Zenodo example | 15.3.2021 at 1-2 PM
- The Open Data Directive and research data licensing | 22.3.2021 at 1-2 PM
- What’s new in publishing? | 30.3.2021 at 1-2 PM | Enrollment (Aalto staff) / Enrollment (Aalto students)
- Let’s get this metadata right! Maximise the reuse, findability and citation of your dataset | 21.4.2021 at 1-2 PM
- Making your software useful | 7.4.2021 at 12-1 PM
- Research ethics for doctoral students

Registration and more details at: https://www.aalto.fi/en/services/rdm-training

Scientific computing

Please check https://mycourses.aalto.fi/ for other courses at Aalto and https://www.csc.fi/en/training for training courses and events at CSC. Some coming courses:

- CS-E4580 Programming Parallel Computers
- CS-E4640 Big Data Platforms D

MOOC on scientific computing:

- https://www.futurelearn.com/courses/python-in-hpc

9.1.3 Skills map

There is a lot to learn, and it all depends on each other. How do you get started?

Our training map Hands-on Scientific Computing sorts the skills you need by level and category, providing you a strategy to get started.

In order to do basic scientific computing, C (Linux and shell) is needed. To use a computer cluster, D (Clusters and HPC) is useful. E (scientific coding) is useful if you are writing your own software.
9.1.4 Our courses

You can browse the material we have developed for our courses by following the links below.

Linux shell tutorial

Course basics

Linux Shell tutorial by Science IT at Aalto University.

Abstract: This course consists of two parts: Linux Shell Basics and Linux Shell Scripting. The first covers introductory level shell usage (which also is a backdoor introduction to Linux basics). The second covers actual BASH scripting, using learning by doing.

Linux Shell Basics: 2 sessions x 3h

Linux Shell Scripting: 4 sessions x 3h, session rough schedule 2x1h25m with 10m break in between.

Setting up instructions for the lecturer: Main terminal white&black with the enlarged font size. One small terminal at the top that shows commands to the learners.

- `export PROMPT_COMMAND='history -a'` # .bashrc or all the terminals one launches commands
- `tail -n 0 -F .bash_history`

Alternatively, `script` allows to follow the session even after sshing to a remote host plus command appear as soon as they are run. The regular expression can be adapted to the lecturer’s PS1, this one assumes `$command`.

- `script -f demos.out # action window`
- `tail -n 1 -f demos.out | while read line; do [[ "$line" =~ \$\ ([^ ]+)\$ ]] && echo ${BASH_REMATCH[1]}; done`

Starred exercises (*) are for advanced users who would like further stimulation.

Based on

- `man bash v4.2` (Triton default version in Feb 2018)
- Advanced BASH scripting guide
- UNIX Power Tools, Shelley Powers etc, 3rd ed, O’Reilly
- common sense and 20+ years Linux experience
- see also other references in the text

PART #1. Linux Shell Basics

1.1 session: processes and files

First touch: getting a BASH shell

Set yourself up with a BASH shell. Connect to a server or open on your own computer. Examples and demos given during the lecture are done on Triton, though should work on all other Linux installations.

- Linux and Mac users: just open a terminal window. If you wish you can login to Triton or any other Aalto Linux server.

---

Windows users: install PuTTY\textsuperscript{2} then $SSH$ to any interactive server at Aalto or your department.

### About the Linux Shell

- A *shell* is what you get when your terminal window is open. It is a command-line (CLI), an interface that interpreters and executes the commands.
- The name comes from being a “shell” (layer) around the operating system. It connects and binds all programs together.
- This is the basic, raw method of using UNIX-like systems. It may not be used everyday, but it’s really good (necessary) for any type of automation and scripting - as is often needed in science, when connecting pieces together, or when using Triton.
- There are multiple shells. This talk is about bash, which is the most common one. zsh is another common shell which is somewhat similar but has some more powerful features. tcsh is another shell from a completely different family (the csh family), which has quite different syntax.
- bash is a “Bourne shell”: the “bourne-again shell”. An open source version of original Bourne shell.
- It may not be obvious, but the concepts here also apply to Windows programs and will help you understand them. They also apply more directly to Mac programs, because Mac is unix under the hood.

### Basic shell operation

- You type things on the screen (standard input or stdin). The shell uses this to make a command.
- The shell takes the command, splits it into words, does a lot more preprocessing, and then runs it.
- When the command runs, the keyboard (still standard input) goes to the process, output (standard output) goes to the screen.

### What’s a UNIX process?

- To understand a shell, let’s first understand what processes are.
- All programs are a process: process is a program in action.
- Processes have:
  - Process ID (integer)
  - Name (command being run)
  - Command line arguments
  - input and output: stdin (input, like from keyboard), stdout (output, like to screen), stderr (like stdout)
  - Return code (integer) when complete
  - Working directory
  - environment variables: key-values which get inherited across processes.
- These concepts bind together *all* UNIX programs, even graphical ones.

Process listing commands (feel free to try, but we play more with them later):

\textsuperscript{2} https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html
You can find info about your user (try them right away):

```
id
echo $SHELL
```

Is your default shell is a `/bin/bash`? Login to kosh/tulita and run `chsh -s /bin/bash`

Another way to find out what SHELL you are running:

```
ps -p $$
```

Where am I: `pwd` (this shows the first piece of process information: current directory)

### Getting help in terminal

Before you Google for the command examples, try:

```
man command_name
```

Your best friend ever – `man` – collection of manuals. Type `/search_word` for searching through the man page. But… if it’s a builtin, you need to use `help`.

### Built-in and external commands

There are two types of commands:

- **shell built-in**: `cd`, `pwd`, `echo`, `test`, etc.
- **external**: `ls`, `date`, `less`, `lpr`, `cat`, etc.
- some can be both: e.g. `test`. Options not always the same!
- For the most part, these behave similarly, which is a good thing! You don’t have to tell which is which.

**Hint** `type -a` to find what is behind the name

- `echo`: prints out `echo something` to type # types whatever you put after

**Disable built-in command** `enable -n echo`, after this `/usr/bin/echo` becomes a default instead of built-in `echo`
Working with processes

All processes are related, a command executed in shell is a child process of the shell. When child process is terminated it is reported back to parent process. When you log out all shell child processes terminated along with the shell. You can see the whole family tree with `ps af`. One can kill a process or make it “nicer”.

```
pgrep -af <name>
kill <PID>
pkill <name>
renice #priority <PID>
```

Making process “nicer”, `renice 19 <PID>`, means it will run only when nothing else in the system wants to. User can increase nice value from 0 (the base priority) up to 19. It is useful when you backup your data in background or alike.

Foreground and background processes

The shell has a concept of foreground and background processes: a foreground process is directly connected to your screen and keyboard. A background process doesn’t have input connected. There can only be one foreground at a time (obviously).

If you add & right after the command will send the process to background. Example: `firefox --no-remote &`, and same can be done with any terminal command/function, like `man pstree &`. In the big picture, the & serves the same role as ; to separate commands, but backgrounds the first and goes straight to the next.

If you have already running process, you can background with `Ctrl-z` and then `bg`. Drawback: there is no easy way to redirect the running task output, so if it generates output it covers your screen.

List the jobs running in the background with `jobs`, get a job back online with `fg` or `fg <job_number>`. There can be multiple background jobs.

Kill a foreground job: `Ctrl-c`  

**Hint:** For running X Window apps while you logged in from other Linux / MacOS make sure you use `ssh -X ...` to log in. For Windows users, you need to install Xming\(^3\) on your workstation.

**Hint:** For immediate job-state change notifications, use `set notify`. To automatically stop background processes if they try writing to the screen use `stty tostop`

Exit the shell and ‘screen’ utility

`logout` or `Ctrl-d` (if you don’t want `Ctrl-d` to quit, set `export IGNOREEOF=1 in .bashrc`).

Of course, quitting your shell is annoying, since you have to start over. Luckily there are programs so that you don’t have to do this. In order to keep your sessions running while you logged out, you should learn about the `screen` program.

- `screen` to start a session
- `Ctrl-a d` to detach the session while you are connected
- `screen` `-ls` to list currently running sessions
- `screen` `-rx <session_id>` to attach the session, one can use TAB for the autocompletion or skip the `<session_id>` if there is only one session running
- `tmux` is a newer program with the same style. It has some extra features and some missing features still.

\(^3\) http://www.straightrunning.com/XmingNotes/
Some people have their screen open forever, which just keeps running and never gets closed. Wherever they are, they ssh in, connect, and resume right where they left off.

Example: `irssi` on kosh / lyta

[Lecture notes: that should be a first half, then joint hands-on/break ~30 mins]

**Exercise 1.1.1**

- for Aalto users: set your SHELL to BASH if you have not yet done so: `chsh -s /bin/bash` on kosh
- find out with `man` how to use `top` / `pstree` / `ps` to list all the running processes that belong to you
  
  Tip: `top` has both command line options and hot keys.
  
  - (*) see `man ps` and find out how to list a processes tree with `ps`, both all processes and only your own (but all your processes, associated with all terminals)
- with `pgrep` list all bash and then zsh sessions on kosh or triton
- log in to triton/kosh and run `man ps`, send it to background, and `logout`, then log in again. Is it still there? Play with the `screen`, run a session, then detach it and log out, then log in back and get your original screen session back.
- run `man htop`, send it to background, and then kill it with `kill`. Tip: one can do it by background job number or by PID.
- Imagine a use case: your current ssh session got stuck and does not response. Open another ssh session to the same remote host and kill the first one. Tip: `echo $$` gives you current bash PID.
  
  - (*) get any X Window application (firefox, xterm, etc) to run on Triton / kosh

**Files and directories**

Files contain data. They have a name, permissions, owner (user+group), contents, and some other metadata.

Filenames may contain any character except ‘/’, which is reserved as a separator between directory and filenames. The special characters would require quotation while dealing, with such filenames, though it makes sense to avoid them anyway.

Path can be absolute, starts with ‘/’ or relative, that is related to the current directory.

`ls` is the standard way of getting information about files. By default it lists your current directory (i.e. `pwd`), but there are many options:

```bash
# list directory content
ls /scratch/work

# list directory files including dot files (i.e. hidden ones)
ls -A ~/directory1

# list all files and directories using long format (permissions, timestamps, etc)
ls -lA ../../directory2
```

Special notations and expansions in BASH, can be used with any command:

```
./, ../, ~, *, ?, [], [!], {abc,xyz}, (1..10)
```

For the quotation:
Quotation matters: ``$USER`` vs echo `$USER`

BASH first expands the expansions and substitute the wildcards, and then execute the command. Could be as complex as:

```
ls -l ~/[!abc]???/dir{123,456}/filename*.[1..9].txt
```

There are a variety of commands to manipulate files/directories:

```
cd, mkdir, cp, cp -r, rm, rm -r, mv, ln, touch
```

For file/directory meta information or content type:

```
ls, stat, file
```

Note that `cd` is a shell builtin which change’s the shell’s own working directory. This is the base from which all other commands work: `ls` by default tells you the current directory, `.` is the current directory, `..` is the parent directory, `~` is your HOME. This is inherited to other commands you run. `cd` with no options drops your to your $HOME.

```bash
# copy a directory preserving all the metadata to two levels up
cp -a dir1/ ../../

# move all files with the names like filename1.txt, filename_abc.txt etc to dir2/
mv filename*.txt dir2/

# remove a directories/files in the current dir without asking for the confirmation
rm -rf dir2/ dir1/ filename*

# create an empty file if doesn't exist or update its access/modification time
touch filename

# create several directories at once
mkdir dir3 dir4 dir5
# -or-
mkdir dir{3,4,5}

# make a link to a target file (hard link by default, -s for symlinks)
ln target_file ../link_name
```

Discover other `ls` features `ls -lx, ls -ltr, ls -Q`

You may also find useful `rename` utility implemented by Larry Wall.

**File/directory permissions**

- Permissions are one of the types of file metadata.
- They tell you if you can **read** a file, **write** a file, and **execute a file/list directory**
- Each of these for both **user**, **group**, and **others**
- Here is a typical permission bits for a file: `-rw-r--r--`
- In general, it is `rwxrwxrwx` – read, write, execute/search for user, group, others respectively
- `ls -1` gives you details on files.
Modifying permissions: the easy part

chmod/chown is what will work on all filesystems:

- `chmod u+rwx,g-rwx,o-rwx <files>`  # u=user, g=group, o=others, a=all
- `chmod 700 <files>`  # r=4, w=2, x=1

# recursive, changing all the subdirectories and files at once
- `chmod -R <perm> <directory>`

# changing group ownership (you must be a group member)
- `chgrp group_name <file or directory>

Extra permission bits:

- s-bit: setuid/setgid bit, preserves user and/or group IDs.
- t-bit: sticky bit, for directories it prevents from removing file by another user (example /tmp)

Setting default access permissions: add to `.bashrc` umask 027. The `umask` is what permissions are removed from any newly created file by default. So `umask 027` means “by default, g-w,o-rwx any newly created files”. It’s not really changing the permissions, just the default the operating system will create with.

**Hint:** even though file has a read access the top directory must be searchable before external user or group will be able to access it. Sometimes on Triton, people do `chmod -R o-rwx $WRKDIR; chmod o+x $WRKDIR`. Execute (x) without read (r) means that you can access files inside if you know the exact name, but not list the directory. The permissions of the files themselves still matter.

Modifying permissions: advanced (*)

Access Control Lists (ACLs) are advanced access permissions. They don’t work everywhere, for example mostly do no work on NFS mounted directories. They are otherwise supported on ext4, lustre, etc (thus works on Triton $WRKDIR).

- In “normal” unix, files have only “owner” and “group”, and permissions for owner/group/others. This can be rather limiting.
- Access control lists (ACLs) are an extension that allows an arbitrary number of users and groups to have access rights to files. The basic concept is that you have:
- ACLs don’t show up in normal `ls -l` output, but there is an extra plus sign: `-rw-rw-rw-`. ACLs generally work well, but there are some programs that won’t preserve them when you copy/move files, etc.
- POSIX (unix) ACLs are controlled with `getfacl` and `setfacl`
  - Allow read access for a user `setfacl -m u:<user>:r <file_or_dir>`
  - Allow read/write access for a group `setfacl -m g:<group>:rw <file_or_dir>`
  - Revoke granted access `setfacl -x u:<user> <file_or_dir>`
  - See current stage `getfacl <file_or_dir>`

File managers on Triton we have installed Midnight Commander – `mc`

Advanced file status to get file meta info `stat <file_or_dir>`

[Lecture notes: hands-on ~30 mins till the end of this session]

---

8 https://www.computerhope.com/unix/uumask.htm
Exercise 1.1.2

- mkdir in your $HOME (or $WRKDIR if on Triton), cd there and touch a file. Rename it. Make a copy and then remove the original. What does touch do?
- list all files in /usr/bin and /usr/sbin that start with non-letter characters with one ls command
- (*) list with ls dot files/directories only (by default it lists all files/directories but not those that begin with .). “dotfiles” are a convention where filenames that begin with . such as .bashrc are considered “hidden”.
- Explore stat file output. What metadata do you find? Try to stat files of different types (regular file, directory, link, special device in /dev, named pipe)
- create a directory, use chmod to allow user and any group members full access and no access for others
- (*) change that directory group ownership with chown or chgrp (any group that you belong to is fine), set s-bit for the group and apply t-bit to a directory, check that the upper directory has o+x bit set: now you should have a private working space for your group. Tip: see groups that you are a member of id -Gn
- ls -ld tells you that directory has permissions rwxr-Sr-. Do group members have access there?
- create a directory (in WRKDIR if on Triton and in /tmp if on any other server), use setfacl to set its permissions so that only you and some user/group of your choice would have access to it.
- (*) create a directory and a subdirectory in it and set their permissions to 700 with one command.

1.2 session: interactive usage

find

- find is a very unixy program: it finds files, but in the most flexible way possible.
- It is a amazingly complicated program
- It is a number one in searching files in shell

With no options, just recursively lists all files starting in current directory:

```
find
```

The first option gives a starting directory:

```
find /etc/
```

Other search options: by modification/accessing time, by ownership, by access type, joint conditions, case-insensitive, that do not match, etc:

```
# -or- 'find ~ $WRKDIR -name file.txt' one can search more than one dir at once
find ~ -name file.txt
```

(continues on next page)

---

5 https://alvinalexander.com/unix/edu/examples/find.shtml
6 http://www.softpanorama.org/Tools/Find/index.shtml
Find syntax is actually an entire boolean logic language given on the command line: it is a single expression evaluated left to right with certain precedence. There are match expressions and action expressions. Thus, you can get amazingly complex if you want to. Take a look at the ‘EXAMPLES’ section in man find for the comprehensive list of examples and explanations.

**find on Triton** On Triton’s WRKDIR you should use lfs find. This uses a raw lustre connection to make it more efficient than accessing every file. It has somewhat limited abilities as comparing to GNU find. For details man lfs on Triton.

**Fast find – locate** Another utility that you may find useful locate <pattern>, but on workstations only. This uses a cached database of all files, and just searches that database so it is much faster.

**Too many arguments** error solved with the find ... | xargs

### File archiving

**tar** is the de-facto standard tool for saving many files or directories into a single archive file. Archive files may have extensions .tar, .tar.gz etc depending on compression.

```bash
# create tar archive gzipped on the way
tar -caf archive_name.tar.gz directory_to_be_archived/

# extract files
tar -xaf archive_name.tar.gz -C path/to/directory
```

Other command line options: *r* - append files to the end of an archive, *t* - list archive content, *f* is for the filename, and *a* selects the compression method based on the archive file suffix (in this example gzip, due to the .gz suffix. Without compression files/directories are simply packed as is.

```bash
# xz has better compression ratio than gzip, but is very slow
tar -caf archive_file.tar.xz dir1/ dir2/
```

Individual files can be compressed directly, e.g. with gzip:

```bash
# file.gz is created, file is removed in the process.
gzip file
```
Transferring files (+archiving on the fly)

For Triton users the ability to transfer files to/from Triton is essential. Same applicable to file transfer between your home workstation and kosh etc.

Several use cases:

```bash
# transferring a file from your HOME on kosh to your home workstation
sftp AALTO_LOGIN@kosh.aalto.fi:file_to_copy

# transferring files from Triton to your Aalto workstation
sftp triton.aalto.fi:/scratch/work/LOGIN_NAME/some/files/* path/to/copy/to
```

(*) Another use case, copying to Triton, or making a directory backup with rsync:

```bash
rsync -urlptDxv --chmod=Dg+s somefile triton.aalto.fi:/scratch/work/LOGIN_NAME #
→ copy a file to $WRKDIR
rsync -urlptDxv --chmod=Dg+s dir1/ triton.aalto.fi:/scratch/work/LOGINNAME/dir1/ #
→ sync two directories
```

(*) Transferring and archiving your Triton data on the fly to some other place:

```bash
# login to Triton
cd $WRKDIR
tar czf - path/to/dir | ssh kosh.aalto.fi 'cat > path/to/archive/dir/archive_file.tar.gz'
```

[Lecture notes: this session has three theory+exercises hands-ons, roughly 40+20 minutes each]

**Exercise 1.2.1**

- Find with `find` all the files in your `$HOME` that are readable or writable by everyone
  - (*) apply `chmod o-rwx` to all recently found files with `find`
- Make a tar.gz archive of any of your directory at your HOME (or WRKDIR if on Triton), when done list the archive content, then append another file/directory to the existing archive.
  - (*) Extract only one particular file to some subdirectory from the archive
- Transfer just created archive using either `sftp` or `rsync`.
  - (*) Try ssh+tar combo to make transfer and archive on the fly.
How to make things faster: hotkeys

- Is it annoying to have to type everything in the shell? No, because we have hotkeys. In fact, it can become much more efficient and powerful to use the shell.
- Most important key: TAB: autocomplete. You should never be typing full filenames or command names. TAB can complete almost anything

Common hotkeys:

- TAB – autocompletion
- Home or Ctrl-a – start of the command line
- End or Ctrl-e – end
- Ctrl-left/right arrows or Alt-b/Alt-f - moving by one word there and back
- up/down arrows – command history
- Ctrl-l – clear the screen
- Ctrl-Shift-c – copy
- Ctrl-Shift-v – paste
- Ctrl-Shift– – undo the last changes on cli
- Alt-r – undo all changes made to this line
- Ctrl-r – command history search: backward (hit Ctrl-r, then start typing the search word, hit Ctrl-r again to go through commands that have the search word in it)
- Ctrl-s – search command history furtherword (for this to work one needs to disable default suspend keys stty -ixon)
- Ctrl-u – remove beginning of the line, from cursor
- Ctrl-k – remove end of the line, from cursor
- Ctrl-w – remove previous word

inputrc Check /etc/inputrc for some default key bindings, more can be defined ~/.inputrc (left as a home exercise)

CDPATH helps changing directories faster. When you type cd dirname, the shell tries to go to one of the local subdirectories and if it is not found shell will try the same command from every directory listed in the $CDPATH.

```
export CPATH=$HOME:$WRKDIR:$WRKDIR/project
```

Initialization files and configuration

- When the shell first starts (when you login), it reads some files. These are normal shell files, and it evaluates normal shell commands to set configuration.
- You can always test things in your own shell and see if it works before putting it in the config files. Highly recommended!
- You customize your environment means setting or expanding aliases, variables, functions.
- The config files are:
  - .bashrc (when SSH) and
  - .bash_profile (interactive login to a workstation)
Aalto scientific computing guide

– they are often a symlink from one to another

• To get an idea how complicated .bashrc can be take a look at <https://www.tldp.org/LDP/abs/html/sample-bashrc.html>

One of the things to play with: command line prompt defined in PS1

```
PS1="[\d \t \u@\h:~\w ] $ 
```

For special characters see PROMPTING at man bash. To make it permanent, should be added to .bashrc like

```
export PS1.
```

Creating/editing/viewing file

• A text editor edits files as ASCII. These are your best friend. In fact, text files are your best friend: rawest, most efficient, longest-lasting way of storing data.

• “pager” is a generic term for things that view files or data.

Linux command line text editors like:

• nano - simplest
• vim - minimal. To save&quit, ESC :wq
• emacs - or the simplest one nano. To save&quit: Ctrl-x Ctrl-c

To view contents of a file in a scrollable fashion: less

Quick look at the text file cat filename.txt (dumps everything to screen- beware of non-text binary files or large files!)

Other quick ways to add something to a file (no need for an editor)

```
echo 'Some sentence, or whatever else 1234567!-+>$#' > filename.txt
cat > filename2.txt to finish typing and write written to the file, press enter, then Ctrl-d.
```

The best text viewer ever less -S (to open a file in your EDITOR, hit v, to search through type /search_word)

Watching files while they grow tail -n 0 -f <file>

Try: add above mentioned export PS1 to .bashrc. Remember source .bashrc to enable changes

Exercise 1.2.2

• link .bash_profile to .bashrc. Tip: see ln command from the previous session.
• open ~/.bashrc for editing and add there CDPATH example from above, customize it for your needs and test. Tip: remember source ~/.bashrc.
• add umask 027 to .bashrc, try creating files. Tip: umask -S prints your current setting.
• customize a prompt $PS1 and add it to your .bashrc, make sure is has a current directory name and the hostname in it in the format hostname:/path/to/current/dir. Hint: save the original PS1 like oldPS1=$PS1 to be able to recover it any time.
• (*) Set some default options for the less program in your bashrc. Examples: case-insensitive searching, long prompt, wrapping lines.

---

Utilities: the building blocks of shell

- wide range of all kind of utilities available in Linux
- shell is a glue to bind them all together
- commandline is often a long list of those utilities joint into pipe that pass output of each other further

| cat; sort; tr; cut; head; date; tail; wc; grep; uniq; paste; find | # and many others |

We catch many of them on the way.

Input and output: redirect and pipes

- Programs can display something: `echo this is some output` or `cat`
- Programs can take some input: e.g. `less` by default displays input if no filename given.
- `cat /etc/bashrc` dumps that file to `standard output` (stdout)
- `cat /etc/bashrc | less` gives it to `less` on `standard input` (stdin)

Pipe: output of the first command as an input for the second one `command_a | command_b`:

```
# send man page to a default printer
man -t ls | lpr

# see what files/directories use the most space, including hidden ones
du -hs *.[!.]* | sort -h

# count a number of logged in users
w -h | wc -l

# to remove all carriage returns and Ctrl-z characters from a Windows file
cat win.txt | tr -d '\15\32' > unix.txt

# to list all matching commands
history | grep -w 'command name'

# print all non-printable characters as well
ls -lA | cat -A

# print the name of the newest file in the directory (non-dot)
ls -ltF | grep -v -E '*/|@' | head -1
```

Redirects:

- Like pipes, but send data to/from files instead of other processes.
- Replace a file: `command > file.txt`
- Append to a file: `command >> file.txt` (be careful you do not mix them up!)
- Redirect file as STDIN: `command < file` (in case program accepts STDIN only)

```
echo Hello World > hello.txt

ls -lH >> current_dir_ls.txt

# join two files into one
```
(continued from previous page)

```
# extract user names and store them to a file
getent passwd | cut -d: -f1,5 > users

# join file1 and 2 lines one by one using : as a delimiter
paste -s -d : file1 file2 > file3

# go through file1 and replace spaces with a new line mark, then output to file2
tr -s ' ' '\n' < file1 > file2

This is the unix philosophy and the true power of the shell. The unix philosophy is a lot of small, specialized, good programs which can be easily connected together. The beauty of the cli are elegant one-liners i.e. list of commands executed in one line.

To dump output of all commands at once: group them.

```
{ command1; command2; } > filename  # commands run in the current shell as a group
( command1; command2; ) > filename  # commands run in external shell as a group
```

Coreutils by GNU You may find many other useful commands at https://www.gnu.org/software/coreutils/manual/coreutils.html

**Pipelines: ;, &&, and ||**

- You can put several commands on the same line using different separators.
- The shell term for this is pipelines.

Chaining: `command_a ; command_b`: always runs both commands.

Remember exit codes? In shell, 0=succes and anything 1-255=failure. Note that this is opposite of normal Boolean logic!

The && and || are short-circuit (lazy) boolean operators. They can be used for quick conditionsals.

- `command_a && command_b`
  - If `command_a` is successful, also run `command_b`
  - Final exit code is last evaluated one, which has the role of Boolean `and`.

- `command_a || command_b`
  - If `command_a` is not successful, also run `command_b`
  - Final exit code is that of the last evaluated command, which has the role of Boolean `or`.

**Hint** `command_a && command_b || command_c`

Try: `cd /nonexistent_dir && ls /nonexistent_dir` compare with `cd /nonexistent_dir; ls /nonexistent_dir`

Try: `ping -c 1 8.8.8.8 > /dev/null && echo online || echo offline`
grep

Later on you’ll find out that grep is one of the most useful commands you ever discover on Linux (except for all the other most useful commands ever)

```
grep <pattern> <filename>  # grep lines that match <pattern>
-or-
command | grep <pattern>  # grep lines from stdin
```

```
# search all the files in the dir/ and its subdirs, to match the word 'is', case insensitive
grep -R -iw 'is' dir/

# grep all lines from *command* output, except those that have 'comment' in it
*command* | grep -v comment

# displaying 2 extra lines before and after the match (-A just after, -B just before)
grep -C 2 'search word' file

# counts the number of matches
grep -c <pattern> file(s)

# shows only the matched part of the string (by default grep shows whole line)
grep -o <pattern> file(s)

# accepts way more advanced regular expressions as a search pattern
grep -E <extended_regexpr> file(s)
```

For details on what <pattern> could be, look for REGULAR EXPRESSIONS at man grep. Some examples:

```
# grep emails to a list
grep -Eio "\b[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,6}\b" file.txt

# grep currently running firefox processes
ps auxw | grep firefox

# grep H1 and H2 header lines out of HTML file
grep "<[Hh][12]>" file.html
```

Exercise 1.2.3

- make a pipe that counts number of filesdirectories (including dot files) in your directory
- grep directories out of ls -l
- grep all but blank lines in triton:/etc/bashrc
  - expand the previous one to filter out commented lines also (line starts with #). Note that lines may have spaces before # mark.
- count unique logged in users on triton. Tip: who users gives you a list of all currently login users, many of them have several sessions open.
- (*) Play with the commands grep, cut: find at least two ways to extract IP addresses only out of /etc/hosts. Tip: grep has -o option, thus one can build a regular expression that will grab exactly what you need.
- (*) Using pipes and commands echo/tr/uniq, find doubled words out of My Do Do list: Find a a Doubled Word. Any easier way to do it?
PART #2. Linux Shell Scripting

Quoting, substitutions, aliases

Last time, we focused on interactive things from the command line. Now, we build on that some and end up with making our own scripts.

Command line processing and quoting

So, shell is responsible for interpreting the commands you type. Executing commands might seem simple enough, but a lot happens between the time you press RETURN and time your computer actually does something.

- When you enter a command line, it is one string.
- When a program runs, it always takes an array of strings (the `argv` in C, `sys.argv` in Python, for example). How do you get from one string to an array of strings? Bash does a lot of processing.
- The simplest way of looking at it is everything separated by spaces, but actually there is more: variable substitution, command substitution, arithmetic evaluation, history evaluation, etc.

The partial order of operations is (don’t worry about exact order: just realize that the shell does a lot of different things in same particular order):

- history expansion
- brace expansion `((1..9))`
- parameter and variable expansion `${VAR}, ${VAR}`
- command substitution `($())`
- arithmetic expansion `${(1+1)}`
- word splitting
- pathname expansion `*, ?, [a,b]`
- redirects and pipes

One thing we will start to see is shell quoting. There are several types of quoting (we will learn details of variables later):

```bash
# Double quotes: disable all other characters except $, '('
echo "$SHELL"
# Single quotes: disable all special characters
echo '$SHELL'
# backslash disables the special meaning of the next character
ls name\ with\ space
```

By special characters we mean:

```bash
# & * ? [ ] ( ) = | ^ ; < > $ " '
```

There are different rules for embedding quoting in other quoting. Sometimes a command passes through multiple layers and you need to really be careful with multiple layers of quoting! This is advanced, but just remember it.
echo 'What's up? how much did you get $$?'  # wrong, ' can not be in between ''
echo "What's up? how much did you get $$?"  # wrong, $$ is a variable in this case
echo "What's up? how much did you get \$$?"  # correct
echo "What's up? how much did you get '"$$'"?"  # correct

At the end of the line \ removes the new line character, thus the command can continue to a next line:

```
ping -c 1 8.8.8.8 > /dev/null && 
echo online || 
echo offline
```

### Substitute a command output

- Command substitutions execute a command, take its stdout, and place it on the command line in that place.

$(command) or alternatively `command`. Could be a command or a list of commands with pipes, redirections, grouping, variables inside. The $() is a modern way, supports nesting, works inside double quotes. To understand what is going on in these, run the inner command first.

```
# 'whoami' alternative
echo $(id -un):$(id -gn)@$(hostname -s)

# save current date to a variable
today=$(date +%Y-%m-%d)

# create a new file with current timestamp in the name (almost unique filename)
touch file.$(date +%Y-%m-%d-%H-%M-%S)

# archive current directory content, where new archive name is based on current path_→and date
tar czf $(basename $(pwd)).$(date +%Y-%m-%d).tar.gz .
```

This is what makes BASH powerful!

Note: $(command || exit 1) will not have an effect you expect, command is executed in a subshell, exiting from inside a subshell, closes the subshell only not the parent script. Subshell can not modify its parent shell environment, though can give back exit code or signal it:

```
# this will not work, echo still will be executed
dir=nonexistent
echo $(ls -l $dir || exit 1)

# this will not work either, since && evaluates echo's exit code, not ls
echo $(ls -l $dir) && exit 1

# this will work, since assignment a comman substitution to a var returns exit # code of the executed command
var=$(ls -l $dir) || exit 1
echo $var
```
More about redirection, piping and process substitution

`STDIN`, `STDOUT` and `STDERR`: reserved file descriptors 0, 1 and 2. They always there whatever process you run. But one can use other file descriptors as well.

File descriptor is a number that uniquely identifies an open file.

`/dev/null` file (actually special operating system device) that discards all data written to it.

```
# discards STDOUT only
command > /dev/null

# discards both STDOUT and STDERR
command &> /dev/null
command > /dev/null 2>&1  # same as above, old style notation

# redirects outputs to different files
command 1>file.out 2>file.err

# takes STDIN as an input and outputs STDOUT/STDERR to a file
command < input_file &> output_file
```

Note, that `&>` and `>&` will do the same, redirect both STDOUT and STDERR to the same place, but the former syntax is preferable.

```
# what happens if 8.8.8.8 is down? How to make the command more robust?
ping -c 1 8.8.8.8 > /dev/null && echo online || echo down

# takes a snapshot of the directory list and send it to email, then renames the file
ls -l > listing && { mail -s "ls -l $(pwd)" jussi.meikalainen@aalto.fi < listing; mv listing listing.$(date +"%Y-%m-%d-%H-%M"); }

# a few ways to empty a file
> filename
cat /dev/null > filename

# read file to a variable
var=$(< path/to/file)

# extreme case, if you can't get the program to stop writing to the file...
ln -s /dev/null filename
```

Pipes are following the same rules with respect to standard output/error. In order to pipe both STDERR and STDOUT `|&`. If `!` preceeds the command, the exit status is the logical negation.

```
# open input_file for reading into the file descriptor 3
exec 3< $input_file

# while open, any command can operate on the descriptor
read -n 3 var <$3
command <$3

# mind the file offset, one can read a line, or a few chars, if you have read the file
```

(continues on next page)
Opening a FD instead of using a file name multiple times may save you some IO. *Hint:* to monitor the file operations (system calls) one may employ `strace -f -c -e trace=write,openat your_script`.

But what if you need to pass to another program results of two commands at once? Or if command accepts file as an argument but not STDIN?

One can always do this in two steps, run commands and save results to file(s) and then use them with the another command. Though BASH helps to make even this part easier (or harder), the feature called *Process Substitution*, looks like `<(command)` or `>(command)`, no spaces in between parentheses and `<` signs. It emulates a file creation out of *command* output and place it on a command line. The *command* can be a pipe, pipeline etc.

The actual file paths substituted are `/dev/fd/<n>`. The file paths can be passed as an argument to the another command or just redirected as usual.

```bash
# BASH creates a file that has an output of *command2* and pass it to *command1*
# file descriptor is passed as an argument, assuming command1 can handle it
command1 <(command2)

# same but redirected (like: cat < filename)
command1 < <(command2)

# in the same way one can substitute results of several commands or command groups
command1 <(command2) <(command3 | command4; command5)

# example: comparing listings of two directories
diff <(ls dir1) <(ls dir2)

# and vice versa, *command1* output is redirected as a file to *command2*
command1 > >(command2)

# essentially, in some cases pipe and process substituion do the same
ls -s | cat
cat <(ls -s)
```
Aliases

- Alias is nothing more than a shortcut to a long command sequence.
- With alias one can redefine an existing command or name a new one.
- Alias will be evaluated only when executed, thus it may have all the expansions and substitutions one normally has on the cli.
- They are less flexible than functions which we will discuss next.

```bash
# your own listing command
alias l='ls -lAF'

# shortcut for checking space usage
alias space='du -hs .[!.*]* | sort -h'

# prints in the compact way login:group
alias me='echo "$(id -un):$(id -gn)"'

# redefine rm
alias rm='rm -i'
alias rm='rm -rf'
```

Aliases go to `.bashrc` and available later by default (really, anywhere they can be read by the shell).

[Lecturer’s notes: about 40 mins joint hands-on session + break]

**Exercise 2.1**

- Define above mentioned `ping ...` command as an alias (you name it) in `~/.bashrc` once you verify it works. Then `source ~/.bashrc` and try the new alias. Tip: any path that starts with `~` means the file or directory is in your HOME.
- Create a directory structure, that has five directories and five subdirs in each directory like `dir1/subdir1, dir1/subdir2, ... dir5/subdir5` with one command. Tip: use Brace expansions and see `mkdir -p ...`
- Use command substitution to create an empty file with the date the in the name, like `file. YYYY-MM-DD.out`. Tip: investigate `date +"..."` output format.
- Create a one-liner with `ls, echo, redirections` etc that takes a file path and says whether this file/directory exists or not. Redirect STDOUT/STDERR to /dev/null. See our `ping -c 8. 8.8.8 ...` as an example.
- Use any of the earlier created files to compare there modification times with `stat -c '%y' filename, diff` and the process substitution.
- (*) Make a one-liner that copies a small dir (small (!), to save time/traffic) from your Triton’s `$WRKDIR` (or any other remote server) and sends confirmation to your email with the directory listing attached but use process substitution instead of saving directory listing to a file. Tip: use examples in the text.
- (*) Using pipes and commands `echo, tr, uniq, find doubled words` out of `My Do Do` list: Find a a Doubled Word.
- (*) Pick up `/scratch/scip/BASH/windows.txt` file and convert it to UNIX format using `tr` and redirects only. Tip: remind first session examples.
- (*) Using `find, duplicate current directory tree` (to some other dir, only tree, no content)
Aalto scientific computing guide

• (*) Join find and grep power and find all the files in /{usr/,}{bin,sbin} that have ‘#!/bin/bash’ in
it
Variables, functions, environment
Your ~/bin and PATH
The PATH is an environment variable. It is a colon delimited list of directories that your shell searches through when
you enter a command. Binaries are at /bin, /usr/bin, /usr/local/bin etc. The best place for your own is ~/bin.:
# add to .bashrc
export PATH="$PATH:$HOME/bin"
# after you have your script written, set +x bit and run it
chmod +x ~/bin/script_name.sh
script_name.sh

You can find where a program is using which or type -a, we recommend the later one:
type -a ls
type -a cd

# a binary
# builtin

Other options:
# +x bit and ./
chmod +x script.sh
./script.sh
# that works if script.sh has #!/bin/bash as a first line
# with no x bit
bash script.sh # this will work even without #!/bin/bash

Extension is optional note that .sh extension is optional, script may have any name
Functions as part of your environment
Alias is a shortcut to a long command, while function is a piece of programming that has logic and can accept input
parameters. Functions can be defined on-the-fly from the cli, or can go to a file. Let us set ~/bin/functions and collect
everything useful there.:
# cd to the directory and lists it at once
# can be run as: lcd <path/to/directory>
lcd() {
cd $1
ls -FlA
}
# in one line, note spaces and ; delimiters
lcd() { cd $1; ls -FlA; }
# -or- in a full format
function lcd { cd $1; ls -FlA; }

By now function has been defined, to run it, one has to invoke it.:
source ~/bin/functions
lcd dir1

The function refers to passed arguments by their position (not by name), that is $1, $2, and so forth:
9.1. Training

405


Functions in BASH have return but it only returns the exit code. Useful in cases where you want to 'exit' the function and continue to the rest of the script. By default functions' variables are in the global space, once chaged in the function is seen everywhere else. local can be used to localize the vars. Compare:

```
var=2; f() { var=3; }; f; echo $var
var=2; f() { local var=3; }; f; echo $var
```

If you happened to build a function in an alias way, redefining a command name while using that original command inside the function, you need to type command before the name of the command, like:

```
rm() { command rm -i "$@"; }
```

Variables

In shell, variables define your environment. Common practice is that environmental vars are written IN CAPITAL: $HOME, $SHELL, $PATH, $PS1, $RANDOM. To list all defined variables printenv. All variables can be used or even redefined. No error if you call an undefined var, it is just considered to be empty:

```
# assign a variable, note, no need for ; delimiter
var1=100 var2='some string'

# calling a variable is just putting a $ dollar sign in a front
echo "var1 is $var1"

# re-assign to another var
var3=$var1

# when appending a variable, it is considered to be a string
var+=<string>/<integer>
var1+=50 # var1 is now 10050
var2+=' more' # var2 is 'some string more'

# we come later to how to deal with the integers (Arithmetic Expanssions $((())) below)
```

There is no need to declare things in advance: there is flexible typing. In fact, you can access any variable, defined or not. However, you can still declare things to be of a certain type if you need to:

```
# assign a variable, note, no need for ; delimiter
var1=100 var2='some string'

# calling a variable is just putting a $ dollar sign in a front
echo "var1 is $var1"

# re-assign to another var
var3=$var1

# when appending a variable, it is considered to be a string
var+=<string>/<integer>
var1+=50 # var1 is now 10050
var2+=' more' # var2 is 'some string more'

# we come later to how to deal with the integers (Arithmetic Expanssions $((())) below)
```

```
# assign a variable, note, no need for ; delimiter
var1=100 var2='some string'

# calling a variable is just putting a $ dollar sign in a front
echo "var1 is $var1"

# re-assign to another var
var3=$var1

# when appending a variable, it is considered to be a string
var+=<string>/<integer>
var1+=50 # var1 is now 10050
var2+=' more' # var2 is 'some string more'

# we come later to how to deal with the integers (Arithmetic Expanssions $((())) below)
```

There is no need to declare things in advance: there is flexible typing. In fact, you can access any variable, defined or not. However, you can still declare things to be of a certain type if you need to:

```
# assign a variable, note, no need for ; delimiter
var1=100 var2='some string'

# calling a variable is just putting a $ dollar sign in a front
echo "var1 is $var1"

# re-assign to another var
var3=$var1

# when appending a variable, it is considered to be a string
var+=<string>/<integer>
var1+=50 # var1 is now 10050
var2+=' more' # var2 is 'some string more'

# we come later to how to deal with the integers (Arithmetic Expanssions $((())) below)
```

There is no need to declare things in advance: there is flexible typing. In fact, you can access any variable, defined or not. However, you can still declare things to be of a certain type if you need to:

```
# assign a variable, note, no need for ; delimiter
var1=100 var2='some string'

# calling a variable is just putting a $ dollar sign in a front
echo "var1 is $var1"

# re-assign to another var
var3=$var1

# when appending a variable, it is considered to be a string
var+=<string>/<integer>
var1+=50 # var1 is now 10050
var2+=' more' # var2 is 'some string more'

# we come later to how to deal with the integers (Arithmetic Expanssions $((())) below)
```
• $? exit status of the last command
• $$ current shell pid
• $# number of input parameters
• $0 running script name, full path
• $FUNCNAME function name being executed, [ note: actually an array ${FUNCNAME[*]} ]
• $1, $2 . . . input parameter one by one (function/script)
• “$@” all input parameters as is in one line

```bash
f() { echo -e " number of input params: $#\n input params: $@
script name: $0\n function name: $FUNCNAME"; return 1; }; f arg1 arg2; echo "exit code: $?"
```

What if you assing a variable to a variable like:

```bash
var2='something'
var1=$var2
echo $var1   # will return '$var2' literally

# BASH provides built-in 'eval' command that reads the string then re-evaluate it
# if variables etc found, they are given another chance to show themselves

eval echo $var1   # returns 'something'
```

In more realistic examples it is often used to compose a command string based on input parameters or some conditionals and then evaluate it at very end.

**Magic of BASH variables**

BASH provides wide abilities to work with the vars “on-the-fly” with ${var...} like constructions. This lets you do simple text processing easily. These are nice, but are easy to forget so you will need to look them up when you need them.

• Assign a $var with default value if not defined: ${var:=value}
• Returns $var value or a default value if not defined: ${var:-value}
• Print an error_message if var empty: ${var:?error_message}
• Extract a substring: ${var:offset:length}, example var=abcde; echo ${var:1:3} returns 'bcd'
• Variable’s length: ${#var}
• Replace beginning part: ${var#prefix}
• Replace trailing part: ${var%suffix}
• Replace pattern with the string: ${var/pattern/string}
• Modify the case of alphabetic characters: ${var,,} for lower case or ${var^^} for upper case

```bash
# will print default_value, which can be a variable
var=''; echo ${var:-default_value}
var1=another_value; var=''; echo ${var:-$var1}
```

(continues on next page)
# assign the var if it is not defined
# note that we use ':' no operation command, to avoid BASH's 'command not found' errors
$: {var:=default_value}

# will print 'not defined' in both cases
var=''; echo ${var:?not defined}
var=''; err='not defined'; echo ${var:?$err}

# will return 'love you'
var='I love you'; echo ${var:2:8}

# will return 15, that is a number of characters
var='I love you too!'; echo ${#var}

# returns file.ext
var=26_file.ext; echo ${var#[0-9][0-9]_}

# in both cases returns 26_file
var=26_file.ext; echo ${var%.ext}
var=26_file.ext; echo ${var%.[a-z][a-z][a-z]}

# returns 'I hate you'
var='I love you'; echo ${var/love/hate}

# other options for substitutions
var=' some text ';
echo ${var/# /} # returns without the first space
echo ${var/% /} # without the last space
echo ${var// /} # without spaces at all

Except for the := the variable remains unchanged. If you want to redefine a variable:

var='I love you'; var=${var/love/hate}; echo $var # returns 'I hate you'

BASH allows indirect referencing, consider:

var1='Hello' var2=var1
echo $var2 # returns text 'var1'
echo ${!var2} # returns 'Hello' instead of 'var1'

To address special characters:

# replacing all tabs with the spaces in the var
var=${var//$'\t'/ }

[LECTURER'S NOTE: ~20 MINUTES FOR THE HANDS-ON EXERCISES. SOLUTION EXAMPLES CAN BE GIVEN AT VERY END.]

Exercise 2.2

- Expand `lcd()` function to have WRKDIR as a default directory in case function is invoked without any input parameter.

- Expand the Exercise 2.1’s `ls ... && echo .. || echo example, make a function that check any file/directory existense given as an argument, like `checkexist path/to/file`. If no argument given, function must return an error message “File or directory not found”.

- Implement a `spaceusage()` function with `du ... | sort ... (see Aliases part examples) that takes directory path as an argument, and if missing uses current directory.
• Using `find` utility, implement a ‘fast find’ function `ff word`. The function should return a long listing (`ls -ldA`) of any file or directory names that contain the `<word>`. Make search case insensitive. Note: your newly created functions should go to `~/bin/functions` file.

• Write two functions `get_filename()` and `get_extension()`. Both should accept a full filename like `path/to/filename.ext` of any length and return `filename` or `ext` correspondingly. Extension can be of any length. Function should handle missing argument case correctly.

• Expand `get_filename()` so that it would accept extension pattern as a second argument (if given) and return `filename` out of `path/to/filename.tar.gz` or alike. I.e. `get_filename path/to/ filename.tar.gz tar.gz`

• (*) By now one should be able to explain: `:(){ :|:& };:`; `. &` in this case sends process to background. [WARNING: it is a forkbomb]

• (*) On Triton write a function that `lfs` find all the dirs/files at `$WRKDIR` that do not belong to your group and fix the group ownership. Use `find ... | xargs`. Tip: on Triton at `$WRKDIR` your username `$USER` and group name are the same. On any other filesystem, `$(id -gn)` returns your group name. One can

• (*) Expand the function above to set group’s s-bit on all the `$WRKDIR` directories.

### Conditionals

#### Tests: `[[ ]]`

• `[[ expression ]]` returns 0=true/success or 1=false/failure depending on the evaluation of the conditional expression.

• `[[ expression ]]` is a new upgraded variation on `test` (also known as `[ ... ]`), all the earlier examples with single brackets that one can find online will also work with double

• Inside the double brackets it performs tilde expansion, parameter and variable expansion, arithmetic expansion, command substitution, process substitution, and quote removal

• Conditional expressions can be used to test file attributes and perform string and arithmetic comparisons

#### Selected examples file attributes and variables testing:

• `-f file` true if is a file
• `-r file` true if file exists and readable
• `-d dir` true if is a directory
• `-e file` true if file/dir/etc exists in any form
• `-z string` true if the length of string is zero (always used to check that var is not empty)
• `-n string` true if the length of string is non-zero
• `file1 -nt file2` true if `file1` is newer (modification time)
• many more others

# checks that file exists
```
[[ -f $file ]] && echo $file exists || { echo error; exit 1; }
```

# check that directory does not exist before creating one
```
[[ -d $dir ]] || mkdir $dir
```
# Check if script/function is given an argument
[[ -z $1 ]]] && { echo no argument; exit 1; }

Note that integers have their own construction (`(( expression ))`) (we come back to this), though `[[ ]]` will work for them too. The following are more tests:

- `==` strings or integers are equal (= also works)
- `!=` strings or integers are not equal
- `string1 < string2` true if `string1` sorts before `string2` lexicographically
- `>` vice versa, for integers greater/less than
- `string =~ pattern` matches the pattern against the string
- `&&` logical AND, conditions can be combined
- `||` logical OR
- `!` negate the result of the evaluation
- `()` group conditional expressions

# Find out were we are
[[ $(pwd) == /some/path ]] ...

# Check, grouping, booleans as a demo
[[ $(hostname -s) == kosh && ($(pwd) == $WORK || $(pwd) == $SCRATCH) ]] ...

# note that `[[ ]]` always require spaces before and after brackets (!)

In addition (old school), double brackets inherit several operands to work with integers mainly:

- `:-eq, -ne, -lt, -le, -gt, -ge` equal to, not equal to, less than, less than or equal to, greater than, or greater than or equal

# Some use cases for `[[ ]]`

# a popular way to check input arguments, if no input, exit (in functions # 'return 1'). Remember, `#$` is special variable for number of arguments.
[[ $# -eq 0 ]] && { echo Usage: $0 arguments; exit 1; }

# if dir exists and is not empty, then do smth
$d=path/to/dir; [[ -d $d && $(ls -A $d) ]] && tar caf ...

# append PATH
d=/path/to/bin; [[ -d $d && $(echo $PATH|grep $d) ]] && export PATH+=:$d

The matching operator `=~` brings more opportunities, because regular expressions come in play. Even more: matched strings in parentheses assigned to `$BASH_REMATCH[[]]` array elements!

- Regular expressions (regexs) are basically a mini-language for searching within, matching, and replacing text in strings.
- They are extremely powerful and basically required knowledge in any type of text processing.
- Yet there is a famous quote by Jamie Zawinski: “Some people, when confronted with a problem, think ‘I know, I’ll use regular expressions.’ Now they have two problems.” This doesn’t mean regular expressions shouldn’t be used, but used carefully. When writing regexes, start with a small pattern and slowly build it up, testing the
matching at each phase, or else you will end up with a giant thing that doesn’t work and you don’t know why and can’t debug it. There are also online regex testers which help build them.

• While the basics (below) are the same, there are different forms of regexs! For example, the grep program has regular regexs, but grep -E has extended. The difference is mainly in the special characters and quoting. Basically, check the docs for each language (Perl, Python, etc) you want to use regexs in.

Selected operators:
• . matches any single character
• ? the preceding item is optional and will be matched, at most, once
• * the preceding item will be matched zero or more times
• + the preceding item will be matched one or more times
• {N} the preceding item is matched exactly N times
• {N,} the preceding item is matched N or more times
• {N,M} the preceding item is matched at least N times, but not more than M times
• [abd], [a-z] a character or a range of characters/integers
• ^ beginning of a line
• $ the end of a line
• () grouping items, this what comes to ${BASH_REMATCH[@]}

```
# match an email
email="jussi.meikalainen@aalto.fi"; regex="(.*)@(.*)"; [[ "email" =~ $regex ]]; echo
→${BASH_REMATCH[@]}

# a number out of the text
txt='Some text with #1278 in it'; regex='#([0-9]+ )'; [[ "$txt" =~ $regex ]] && echo $ →{BASH_REMATCH[1]} || echo do not match

# case insensitive matching
var1=ABCD, var2=abcd; [[ ${var1,,} =~ ${var2,,} ]] && ...
```

For case insensitive matching, alternatively, in general, set shopt -s nocasematch (to disable it back shopt -u nocasematch)

**Conditionals: if/elif/else**

Yes, we have [[ ]], && ... || ... but scripting style is more logical with if/else construction:

```
if condition; then
  command1
elif condition; then
  command2
else
  command3
fi
```

At the condition place can be anything what returns an exit code, i.e. [[ ]], command/function, an arithmetic expression ${(( ))}, or a command substitution.
# to compare two input strings/integers
if [[ "$1" == "$2" ]]
then
  echo The strings are the same
else
  echo The strings are different
fi

# checking command output
if ping -c 1 8.8.8.8 &> /dev/null; then
  echo Online
elif ping -c 1 127.0.0.1 &> /dev/null; then
  echo Local interface is down
else
  echo No external connection
fi

# check input parameters
if [[ $# == 0 ]]; then
  echo Usage: $0 input_arg
  exit 1
fi

... the rest of the code

Expanding tarit.sh to a script

#!/bin/bash

# usage: tarit.sh <dirname>

d=$1

# if dirname is given, we archive it
if [[ -d $d ]]; then
  tar caf $(basename $d).$(date +%Y-%m-%d).tar.gz $d
elif [[ -z $d ]]; then
  tar caf $(basename $(pwd)).$(date +%Y-%m-%d).tar.gz .
else
  echo $d does not exist
fi

case

Another option to handle flow, instead of nested if's, is case.

read -p "Do you want to create a directory (y/n)? " yesno  # expects user input

 case $yesno in
 y|yes)
   dir='dirname'
   echo Creating a new directory $dir
   mkdir $dir
   cd $dir
   ;;
 n|no)
   echo Proceeding in the current dir $(pwd)

(continues on next page)
In the example above, we introduce `read`, a built-in command that reads one line from the standard input or file descriptor.

`case` tries to match the variable against each pattern in turn. Understands patterns rules like `*`, `?`, `[]`, `|`.

Here is the `case` that could be used as an idea for your `~/.bashrc`:

```bash
host=$(hostname)
case $host in
  myworkstation*)
    export PRINTER=mynearbyprinter
    # making your prompt smiling when exit code is 0 :) 
    PS1='$(if 
      [[ -z $WRKDIR ]] && alias cwd="cd $WRKDIR" && cwd
      ;
    triton*)
      [[ -n $WRKDIR ]] && alias cwd="cd $WRKDIR" && cwd
    ;;
    kosh*[brute*|force*]
      PS1='"\u@\h:\w\$'
      export IGNOREEOF=0
    ;;
    *aalto.fi)
      kinit
    ;;
    *)
      echo 'Where are you?'
    ;;
  esac

# is important, if replaced with `;`, execution will continue with the command associated with the next pattern, without testing it. `;;` causes the shell to test next pattern. The default behaviour with `;;` is to stop matches after first pattern has been found.
```

The following example is useful for Triton users: `array jobs`, where one handles array subtasks based on its index.

**Exercise 2.3**

```
# create a file 'cx'
case "$0" in
  *cx) chmod +x "$0" ;;
  *cw) chmod +w "$0" ;;
  *c-w) chmod -w "$0" ;;
  *) echo "$0: seems that file name is somewhat different"; exit 1 ;;
esac

# chmod +x cx
# ln cx cw
# ln cx c-w
# to make a file executable 'cx filename'
```
• Re-implement the above mentioned example...

```bash
[[ -d $d && ! $(echo $PATH|grep $d) ]] ... with the matching operator =~
```

• Improve the tarit.sh script we developed recently:
  
  – add check for the number of the given arguments. Hint: $# must be zero or one.
  
  – validate the given path like path/to/file. Hint: [[ $d =~ regexpr ]], the path may have only alphanumeric symbols, dots, underscore and slashes as a directory delimiter.

• Expand cx script:
  
  – check that $@ not empty
  
  – add option for cr that would add read rights for all. Hint: chmod a+r ...

• (*) Write a function (add to bin/functions) that validates an IPv4 using =~ matching operator. The function should fail incorrect IPs like 0.1.2.3d or 233.204.3.257. The problem should be solved with the regular expression only. Use return command to exit with the right exit code.

**Loops**

**Arithmetic**

BASH works with integers only (no floating point) but supports wide range of arithmetic operators using arithmetic expansion $(( expression )).

• All tokens in the expression undergo parameter and variable expansion, command substitution, and quote removal. The result is treated as the arithmetic expression to be evaluated.

• Arithmetic expansion may be nested.

• Variables inside double parentheses can be without a $ sign.

• BASH has other options to work with the integers, like let, expr, $[], and in older scripts/examples you may see them.

Available operators:

• `n++`, `n--`, `++n`, `--n` increments/decrements

• `+`, `-` plus minus

• `**` exponent

• `*`, `/`, `%` multiplication, (truncating) division, remainder

• `&&`, `||` logical AND, OR

• `expr?expr:expr` conditional operator (ternary)

• `==`, `!=`, `<`, `>`, `>=`, `<=` comparison

• `=`, `+=`, `-=`, `/=`, `%=` assignment

• `()` sub-expressions in parentheses are evaluated first

• The full list includes bitwise operators, see `man bash` section `ARITHMETIC EVALUATION`.

```bash
n=10; ((n++)) # without dollar sign value is not returned, though 'n' has been incremented

# but if we need a value

(continues on next page)```
n=10; m=3; q=$((n**m))

# here we need exit code only
if ((q%2)); then echo odd; fi
if ((n>=m)); then ...; fi

# condition ? integer_value_if_true : integer_value_if_false
n=2; m=3; echo $((n<m?10:100))

# checking number of input parameters, if $# is zero, then exit
# (though the alternative [[ $# == 0 ]] is more often used, and intuitively more,
# clearer)
if ! (($#)); then echo Usage: $0 argument; exit1; fi

# sum all numbers from 1..n, where n is a positive integer
# Gauss method, summing pairs
if (($#==1)); then
  n=$1
else
  read -p 'Give me a positive integer ' n
fi
echo Sum from 1..$n is $((n*(n+1)/2))

Left for the exercise: make a summation directly 1+2+3+...+n and compare performance with the above one.

For anything more mathematical than summing integers, one should use something else, one of the option is bc, often installed by default.

```
# bc -- an arbitrary precision calculator language
# compute Pi number
echo "scale=10; 4*a(1)" | bc -l
```

For loops

BASH offers several options for iterating over the lists of elements. The options include

- Basic construction for arg in item1 item2 item3 ...
- C-style for loop for ((i=1; i <= LIMIT ; i++))
- while and until constructs

Simple loop over a list of items:

```
# note that if you put 'list' in quotes it will be considered as one item
for dir in dir1 dir2 dir3/subdir1; do
  echo "Archiving $dir ..."
  tar -caf ${dir///.}.tar.gz $dir && rm -rf $dir
done
```

If path expansions used (*, ?, [], etc), loop automatically lists current directory:

```
# example below uses ImageMagick's utility to convert all *.jpg files
# in the current directory to *.png.
# i.e. '*.jpg' similar to 'ls *.jpg'
for f in *.jpg; do
```

(continues on next page)
convert "f" "$(f/../*.png)"  # quotes to avoid issues with the spaces in the name done

# another command line example renames *.JPG and *.JPEG files to *.jpg
# note: in reality one must check that a newly created *.jpg file does not exist
for f in *.JPG *.JPEG; do mv -i "$f" "${f/../*.jpg}"; done

# do ... done in certain contexts, can be omitted by framing the command block within
# curly brackets
# and certain for loop can be written in one line as well
for i in {1..10}; { echo i is $i; }

If in list omitted, for loop goes through script/function input parameters $@

# here is a loop to rename files which names are given as input parameters
# touch file{1..3}; ./newname file1 file2 file3 for old; do
  read -p "old name $old, new name: " new
  mv -i "$old" "$new"
done

Note: as side note, while working with the files/directories, you will find lots of examples where loops can be emulated
by find ... -print0 | xargs -0 ...

Loop output can be piped or redirected:

# loop other all Triton users to find out who has logged in within last month
for u in $(getent group triton-users | cut -d: -f4 | tr ',' ' '); do
  echo $u: $(last -Rw -n 1 $u | head -1)
done | sort > filename

The list can be anything what produces a list, like Brace expansion {1..10}, command substitution etc.:

# on Triton, do something to all pending jobs based on squeue output
for jobid in $(squeue -h -u $USER -t PD -o %A); do
  scontrol update JobId=$jobid StartTime=now+5days
done

# using find to make a list of files to deal with; the benefit here is that you work
# with the filename as a variable, which gives you flexibility as comparing to
# 'find ... -exec {}' or 'find ... print0 | xargs -0 ...
for f in $(find . -type f -name '*.sh'); do
  if ! bash -n $f >/dev/null; then
    mv $f $f/./.fixme
  fi
done

C-style, expressions evaluated according to the arithmetic evaluation rules:

N=10
for ((i=1; i <= N ; i++))  # LIMIT with no $
do
  echo -n "$i "
done

Loops can be nested.
While/until loops

Other useful loop statement are while and until. Both execute continuously as long as the condition returns exit status zero/non-zero correspondingly.

```bash
while condition; do
    command1
    command2
    ...
done

# sum of all numbers 1..n; n expected as an argument
n=$1 i=1
until ((i > n)); do
    ((s+=i)); ((i++))
done
echo Sum of 1..$n is $s

# endless loop, can be stopped with Ctrl-c or killed
# drop an email every 10 minutes about running jobs on Triton
# can be used in combination with 'screen', and run in background
while true; do
    squeue -t R -u $USER | mail -s 'running jobs' mister.x@aalto.fi
    sleep 600
done

# with the help of 'read var' passes file line by line,
# IFS= variable before read command to prevent leading/trailing
# whitespace from being trimmed
input='/path/to/txt/file'
while IFS= read -r line; do
echo $line
done < "$input"

# reading file fieldwise, IFS= is a delimiter, note quoting with \
# file='/path/to/file.csv'
while IFS=';\' read -r f1 f2 f3 f4; do
    printf 'Field1: %s, Field2: %s, Field4: %s\n' "$f1" "$f2" "$f4"
done < "$file"

# process substitution
while IFS= read -r line; do
    # do something with the lines
done < <(file -b *)
# instead, one can mistakenly try 'file -b * | while read line; do ... done'
# with pipe, 'while' body will be run in a subshell, and thus all variables
# used inside the loop will die when loop is over
```

All the things mentioned above for for loop applicable to while / until loops.

printf should be familiar to programmers, allows formatted output similar to C printf.9

9 https://wiki.bash-hackers.org/commands/builtin/printf
Loop control

Normally for loop iterates until it has processed all its input arguments. while and until loops iterate until the loop control returns a certain status. But if needed, one can terminate loop or jump to a next iteration.

- **break** terminates the loop
- **continue** jump to a new iteration
- **break n** will terminate n levels of loops if they are nested, otherwise terminated only loop in which it is embedded. Same kind of behaviour for **continue n**.

Even though in most of the cases you can design the code to use conditionals or alike, break and continue certainly add the flexibility.

```bash
# here we expand an earlier example to avoid errors in case $f is missing/not accessible
for f in *.JPG *.JPEG; do
  [[ -r "$f" ]] || { echo "$f is missing on inaccessible"; continue; }
  mv -i "$f" "${f/.*/.jpg}"
done
```

Exercise 2.4

- Expand *tarit.sh* so that it would accept none or multiple directories.
- Using for loop rename all the files with the .txt extension to .fixed.txt. Tip: create dummy .txt files with mkdir d{1..3}; touch d{1..3}/d{1..3}.txt. Tip #2: combine ‘for’ loop with ‘find’: for f in $(find . -name '*.txt'); ....
- Use the while example with .csv, take the *demospace/Finnish_Univ_students_2018.csv* to count total number of students around Finland. Tip: add checking that the number of students field is a number [[ $totalnmb =~ ^[0-9]+$ ]]
- Using built-in arithmetic write a script *daysstill.sh* that counts a number of days till a deadline (vacation/salary). Script takes date as an argument, date format suitable to date -d like days_till 2019-6-1. Tip: use date +%s and convert seconds to days (roughly).
- Make script that takes a list of files and checks if there are files in there with the spaces in the name, and if there are, rename them by replacing spaces with the underscores. Use BASH’s builtin functionality only.
  - As a study case, compare it against find . -depth -name ‘* *’ -execdir rename ’s/ /_/g' {} \;
- Write separate scripts that count a sum of any 1+2+3+4+..+n sequence, both the Gauss version (see above) and summation with **for** { (... )}. Where n is an argument, like gauss.sh 1000. Benchmark them with time for n=10000 or more.
  - (*) Implement both methods within one script as functions and benchmark them within the file
  - (*) For the direct summation one can avoid loops, how? Tip: discover eval $(echo {1..$n})
- (*) Get familiar with the *getent* and *cut* utilities. Join them with a loop construction to write a *mygetentgroup* script or just a oneliner that generates a list of users and their real names that belong to a given group. Like:
$ mygetentgroup group_name
meikalaj1: Jussi Meikäläinen
meikalaml: Maija Meikäläinen
...``

• (*) To Aalto users: on kosh/lyta run net ads search samaccountname=$USER
  accountExpires 2>/dev/null to get your account expiration date. It is a 18-
digit timestamp, the number of 100-nanoseconds intervals since Jan 1, 1601 UTC. Im-
plement a function that accept a user name, and if not given uses current user by de-
fault, and then converts it to the human readable time format. Tip: http://meinit.nl/
  convert-active-directory-lastlogon-time-to-unix-readable-time

  – Expand it to handle “Got 0 replies” response, i.e. account name not found.

Arrays, input, Here Documents

Arrays

BASH supports both indexed and associative one-dimensional arrays. Indexed array can be declared with declare
  -a array_name, or first assignment does it automatically (note: indexed arrays only):

```
arr=(my very first array)
arr=('Otakaari 1' Espoo 02150 [6]='PL 11000')
arr[5]=AALTO
```

To access array elements (the curly braces are required, unlike normal variable expansion):

```
# elements one by one
echo ${arr[0]} ${arr[1]}

# array values at once
${arr@[1]}

# indexes at once
${!arr@[1]}

# number of elements in the array
${#arr@[1]}

# length of the element number 2
${#arr[2]}

# to append elements to the end of the array
arr+=('$value')

# assign a command output to array
arr=$(command)

# emptying array
arr=

# to destroy, delete an array
unset arr

# to unset a single array element
```
unset arr[6]

# sorting array
IFS='\n' sorted=($(sort <<<"${arr[*]}"))

# array element inside arithmetic expansion requires no ${}
(arr[$i]++)

# split a string like 'one two three etc' or 'one,two,three,etc' to an array
# note that IFS=' , ' means that separator is either space or comma, not a sequence of _→_them
IFS=' , ' read -r -a arr <<< "$string"

# splitting a word to an array letter by letter
word=qwerty; arr=($(echo $word | grep -o .))

Loops through the indexed array:

for i in ${!arr[@]}; do
  echo arr[$i] is ${arr[$i]}
done

Negative index counts back from the end of the array, [-1] referencing to the last element.

Quick ways to print array with no loop:

# with keys, as is
declare -p arr

# indexes -- values
echo ${!arr[@]} -- ${arr[@]}

# array elements values one per line
printf "%s\n" "${arr[@]}"

Passing an array to a function as an argument could be the use case when you want to make it local:

f() {
  local arr=(${!1}) # pass $1 argument as a reference
  # do something to array elements
  echo ${arr[@]}
}

# invoke the function, huom that no changes have been done to the original arr[@]
arr=(....)
f arr[@]

BASH associative arrays (this type of array supported in BASH since version 4.2) needs to be declared first (!)
declare -A asarr.

Both indexed arrays and associative can be declared as an array of integers, if all elements values are integers
declare -ia array or declare -iA. This way element values are treated as integers always.

asarr=(['university']='Aalto University' ['city']='Espoo ['street address']='Otakaari 1')
asarr[post_index]=02150

Addressing is similar to indexed arrays:
for i in "${!asarr[@]}"; do
  echo asarr[$i] is ${asarr[$i]}
done

Even though key can have spaces in it, quoting can be omitted.

# use case: your command returns list of lines like: 'string1 string2'
# adding them to an associative array like: [string1]=string2
declare -A arr
for i in $(command); do
  arr+=(["${i/ */}"="${i/* /}"])
done

Variable expansions come out in the new light:

# this will return two elements of the array starting from number 1
${arr[@]:1:2}

# all elements without last one
${arr[@]:0:${#arr[@]):-1}

# parts replacement will be applied to all array elements
declare -A emails=([Vesa]=vesa@aalto.fi [Kimmo]=kimmo@helsinki.fi [Anna]=anna@math.tut.fi)
echo ${emails[@]/@*/@gmail.com}
# returns: vesa@gmail.com anna@gmail.com kimmo@gmail.com

For a sake of demo: let us count unique users and their occurances (yes, one can do it with 'uniq -c'.)

# declare associative array of integers
declare -iA arr
for i in $(w -h | cut -c1-8); do # get list of currently logged users into loop
  for u in ${!arr[@]}; do # check that they are unique
    if [[ $i == $u ]]; then
      ((arr[$i]++))
      continue 2
    fi
  done
  arr[$i]=1 # if new, add a new array element
done
for j in ${!arr[@]}; do # printing out
  echo ${arr[$j]} $j
done

Another working demo: script that automates backups or just makes a sync of data to a remote server. Same can be adapted to copy locally, to a usb drive or alike.

# array of directories to be backuped, to skip one, just comment with #
declare -A dirs
dirs[wlocal]=/l/$USER
dirs[xpproject]=/m/phys/extra/project/xp
dirs[homebin]=$HOME/bin
cmd='/usr/bin/rsync' # rsync
args="-auW --delete --progress $@" # accept extra args, like '-n' for the dryrun
...
serv='user@server:backups'  # copying to ~/backups that must exist

# array key is used for the remote dir name
for d in ${!dirs[@]}; do
echo "Syncing ${dirs[$d]}..."
$cmd $args ${dirs[$d]}/ $serv/$d
done

Exercise 2.5

• make a script/function that produces an array of random numbers, make sure that numbers are unique. Print the array nicely using printf for formatting.
  – one version should use BASH functionality only (Tip: $RANDOM)
  – the other one can use shuf

• (*) Pick up the ipvalid function that we have developed earlier, implement IP matching regular expression as ^([0-9]{1,3})\.(\[0-9]{1,3})\.(\[0-9]{1,3})\.(\[0-9]{1,3})$ and work with the ${BASH_REMATCH[*]} array to make sure that all numbers are in the range 0-255

Working with the input

User input can be given to a script in three ways:

• as command arguments, like ./.script.sh arg1 arg2 ...
• interactively from keyboard with read command
• as standard input, like command | ./.script

Nothing stops from using a combination of them or all of the approaches in one script. Let us go through the last two first and then get back to command line arguments.

read can do both: read from keyboard or from STDIN

# the command prints the prompt, waits for the response, and then assigns it to variable(s)
read -p 'Your names: ' firstn lastn

# read into array, each word as a new array element ('arr' declared automatically)
read -a arr -p 'Your names: ' 

Given input must be checked (!) with a pattern, especially if script creates directories, removes files, sends emails based on the input.

# request a new directory name till correct one is given (interrupt with Ctrl-C)
regexp='^[a-zA-Z0-9-/\-]+$'
until [[ "$newdir" =~ $regexp ]]; do
  read -p 'New directory: ' newdir
done

read selected options

• -a <ARRAY> read the data word-wise into the specified array <ARRAY> instead of normal variables
• -N <NCHARS> reads <NCHARS> characters of input, ignoring any delimiter, then quits
• -p <PROMPT> the prompt string <PROMPT> is output (without a trailing automatic newline) before the read is performed
• -r raw input - disables interpretation of backslash escapes and line-continuation in the read data
• -s secure input - don’t echo input if on a terminal (passwords!)
• -t <TIMEOUT> wait for data <TIMEOUT> seconds, then quit (exit code 1)

read is capable of reading STDIN, case like command | ./script, with while read var it goes through the input line by line:

```bash
# IFS= is empty and echo argument in quotes to make sure we keep the format
# otherwise all spaces and new lines shrunk to one and leading/trailing whitespace trimmed

while IFS= read -r line; do
  echo "line is $line"  # do something useful with $line
done

# To check current $IFS
cat -A <<<"$IFS"
```

Though in general, whatever comes from STDIN can be proceeded as:

```bash
# to check that STDIN is not empty
if [[ -p /dev/stdin ]]; then
  # passing STDIN to a pipeline (/dev/stdin can be omitted)
  cat /dev/stdin | cut -d' ' -f 2,3 | sort
fi
```

Other STDIN tricks that one can use in the scripts:

```bash
# to read STDIN to a variable, both commands do the same
var=$(</dev/stdin)
var=$(cat)
```

In the simplest cases like ./script arg1 arg2 ..., you check $# and then assign $1, $2, ... the way your script requires.

```bash
# here we require exactly two arguments
if (($#==2)); then
  var1=$1 var2=$2
  # ... do something useful
else
  echo 'Wrong amount of arguments'
  echo "Usage: ${0##*/} arg1 arg2"
  exit 1
fi
```

To work with all input arguments at once we have $@:

```bash
# $# is a number of arguments on the command line, must be non-zero
if (($#)); then
  for i; do
    echo "$i"
    # ... do something useful with each element of $@
    # note that 'for...' uses $@ by default if no other list given with 'in ...'
  done
else
```

(continues on next page)
Aalto scientific computing guide

As a use case, our tarit.sh script. The script can accept STDIN and arguments, so we check both:

```bash
# Usage: tarit.sh [dirname1 [dirname2 [dirname3 ...]]]
# or command | tarit.sh
# by default no directories to archive. i.e. current
args=''

# checking for STDIN, if any, assigning STDIN to $args
[[ -p /dev/stdin ]] && args=$(</dev/stdin)

# if arguments are given, appending the $args with $@
(( $# )) && args+=" $@

# no arguments, no stdin, then it is a current dir
[[ -z "$args" ]] && args="$(pwd)"

# by now we should have a directory list in $args to archive
for d in $args; do
  # checking that directory exists, if so, archive it
  if [[ -d "$d" ]]; then
    echo Archiving $d ...
    tar caf ${d##*/}.$(date +%Y-%m-%d).tar.gz "$d"
  else
    echo "$d does not exist, skipping."
  fi
done
```

Often, the above mentioned ways are more than enough for simple scripts. But what if options and arguments are like `.script [-f filename] [-z] [-b] [arg1 [arg2 [...] ] ]` or more complex? (common notation: options in the square brackets are optional). What if you write a production ready script that will be used by many other as well?

It is where `getopt` offers a more efficient way of handling script’s input options. In the simplest case `getopt` command (do not get confused with `getopts` built-in BASH function of similar kind) requires two parameters to work: first is a list of valid input options – sequence of letters and colons. If letter followed by a colon, the option requires an argument, if followed by two colons, argument is optional. For example, the string `getopt "sdf:"` says that the options -s, -d and -f are valid and -f requires an argument, like `-f filename`. The second argument required by `getopt` is a list of input parameters (options + arguments) to check, i.e. just `$@`.

Let us use `cx` script as a demo:

```bash
# common usage function with the exit at the end
usage() {
  echo "Usage: $sname [options] file [file [file...]]"
  echo '  -a, gives access to all, like a+x, by default +x'
  echo '  -d <directory/path/bin>, path to the bin directory'
  echo "    can be used in 'cx' to copy a new script there"
  echo '  -a, gives access to all, like a+x, by default +x'
  echo '  -d <directory/path/bin>, path to the bin directory'
  echo "    can be used in 'cx' to copy a new script there"
  echo '  -v, verbose mode for chmod'
  echo '  -h, this help message'
  exit 1
}
# whole trick is in this part: getopt validates the input parameters,  
# structures them by dividing options and arguments with --,  
# and returns them to a variable  
# then they are reassigned back to $@ with 'set --'  
# opts=$(getopt "avhd:" "$@") || usage  
set -- $opts  
# defining variables' default values  
ALL=''  
CMD='/usr/bin/chmod'  
sname=${0##*/} # the name this script was called by  
  
  # by now we have a well structured $@ which we can trust.  
  # to go through options one by one we start an endless 'while' loop  
  # with the nested 'case'. 'shift' makes another trick, every time  
  # it is invoked it is equal to 'unset $1', thus $@ arguments are  
  # "shifted down", $2 becomes $1, $3 becomes $2, etc  
  # 'getopt' adds -- to $@ which separates valid options and the rest  
  # that did not qualify, when it comes to '--' we 'break' the loop  
  # while true; do  
  # case $1 in  
  #   h) usage ;; # output help message and exit  
  #   a) ALL=a ;; # if -a is given we set ALL  
  #   v) CMD+=' -v' ;; # if verbose mode required  
  #   d) shift # shift to take next item as a directory path for -d  
  #      BINDIR=$1  
  #      if [[ -z "$BINDIR" || ! -d "$BINDIR" ]]; then  
  #        echo "ERROR: the directory does not exist"  
  #        usage  
  #      fi  
  #      ;;  
  #   --) shift; break ;; # remove --  
  # esac  
  # shift  
  # done  
  
  # script body  
  case "$sname" in  
  cx*) $CMD $(ALL)+rx "$@" &&  
  [[ -n "$BINDIR" ]] && cp -p $@ $BINDIR ;;  
  cw*) $CMD $(ALL)+w "$@" ;;  
  cr*) $CMD $(ALL)+r "$@" ;;  
  c-w*) $CMD $(ALL)-w "$@" ;;  
  *) echo "ERROR: no idea what $sname is supposed to do"; exit 1 ;;  
  esac  
  
getopt can do way more, go for man getopt for details, as an example:  

# here is getopt sets name with '-n' used while reporting errors: our script name  
# accepts long options like '--filename myfile' along with '-f myfile'  
getopt -n $(basename $0) -o "hac::f:" --long "help, filename:, compress:::" -- "$@"

### Exercise 2.6

* Using the latest tarit.sh (see lecture notes) version as an example, expand above cx script to
accept STDIN, like command | cx [options], where command produces a list of files. Example find . -t file -name '*.sh' | cx -a -d /path/to/bin.

- Using cx demo as an example, expand the latest version of our tarit.sh (see lecture notes) to make it accepting the following options and arguments: tarit.sh -h -y -d <directory/with/backups> [dirname1 [dirname2 [dirname3 ...]]]. By default, with no args, it still should make an archive of the current directory. -h returns usage info, -d <directory/path/with/backups> is a directory the tar archives will go to, your script has to check that directory exists, the script must also check whether a newly created archive already exist and if so, skip creating the archive with the corresponding warning message.

- (*) -y should force overwriting already existing archive.
- (*) -s should make script silent, so that no errors or other messages would come from any inline command.

---

**Here Document, placeholders**

A ‘here document’ and ‘here string’ take the line(s) following and send them to standard input. It’s a way to send larger blocks to stdin.

```bash
# instead of 'echo $STRING | command ...'
command <<<$STRING

# instead of 'cat file | command ...'
command <<<SomeMagicStopWord
The benefit is that one can use $var, ${()} etc in the text
The text ends with the Stop Word on a new line, the word can be any SomeMagicStopWord
```

Often used for messaging, be it an email or dumping bunch of text to file:

```bash
# NAME, SURNAME, EMAIL, DAYS are set earlier

mail -s 'Account expiration' $EMAIL<<END-OF-EMAIL
Dear $NAME $SURNAME,

your account is about to expire in $DAYS days.

$(date)

Best Regards,
Aalto ITS
END-OF-EMAIL
```

Or just outputting to a file (same can be done with echo commands):

```bash
cat <<<EOF >filename
... text
EOF
```

One trick that is particularly useful is using this to make a long comment:

```bash
: <<<\COMMENTS
here come text that is seen nowhere
```

(continues on next page)
there is no need to comment every single line with # COMMENTS

Hint \<<LimitString to turn off substitutions and place text as is with $ marks etc

In case you have a template file which contains variables as placeholders, replacing them:

```bash
# 'template' file like:
The name is $NAME, the email is $EMAIL

# command to substitute the placeholders and redirect to 'output' file
# the original 'template' file remains as is
NAME=Jussi EMAIL=jussi@gmail.com
cat template | while IFS= read -r line; do eval echo $line; done > output
# resulting file: The name is Jussi, the email is jussi@gmail.com
```

**Traps, debugging, profiling**

**Catching kill signals: trap**

What if your script generates temp file and you’d like to keep it clean even if script gets interrupted at the execution time?

The built-in `trap` command lets you tell the shell what to do if your script received signal to exit. It can catch all, but here listed most common by their numbers. Note that signals are one of the common ways of communicating with running processes in UNIX: you see these same numbers and names in programs like `kill`.

- 0 EXIT exit command
- 1 HUP when session disconnected
- 2 INT interrupt - often Ctrl-c
- 3 QUIT quit - often Ctrl-
- 9 KILL real kill command, it can’t be caught
- 15 TERM termination, by `kill` command

```bash
# 'trap' catches listed signals only, others it silently ignores
# Usage: trap group_of_commands/function list_of_signals

trap 'echo Do something on exit' EXIT

# Expanding the backup script from the Arrays section, this can be added to the very beginning:

interrupted() {
    echo 'Seems that backup has been interrupted in the middle'
    echo 'Rerun the script later to let rsync to finish its job'
    exit 1
}

trap interrupted 1 2 15
# ... the rest of the script
```

In other situation, instead of `echo`, one can come up with something else: removing temp files, put something to the log file or output a valuable error message to the screen.
Aalto scientific computing guide

Hint About signals see Standard signals section at man 7 signal. Like Ctrl-c is INT (aka SIGINT).

Debugging and profiling

BASH has no a debugger, but there are several ways to help with the debugging

Check for syntax errors without actual running it bash -n script.sh

Or echos each command and its results with bash -xv script.sh, or even adding options directly to the script. -x enables tracing during the execution, -v makes bash to be verbose. Both can be set directly from the command line as above or with set -xv inside the script.

```
#!/bin/bash -xv
```

To enable debugging for some parts of the code only:

```
set +x
... some code
set -x
```

If you want to check quickly a few commands, with respect to how variables or other substitutions look like, use DEBUG variable set to echo.

```
#!/bin/bash
$DEBUG command1 $arguments
command2
# call this script like 'DEBUG=echo ./script.sh' to see how *command1* looks like
# otherwise the script can be run as is.
```

One can also trap at the EXIT, this should be the very first lines in the script:

```
end() { echo Variable Listing: a = $a b = $b; }
trap end EXIT # will execute end() function on exit
```

For a sake of profiling one can use PS4 and date (GNU version that deals with nanoseconds). PS4 is a built in BASH variable which is printed before each command bash displays during an execution trace. The first character of PS4 is replicated multiple times, as necessary, to indicate multiple levels of indirection. The default is +. Add the lines below right after ‘#!/bin/bash’

```
# this will give you execution time of each command and its line number
# \011 is a tab
PS4='+\011$(date "+%s.%N")\011${LINENO}\011'
set -x
```

Optionally, if you want tracing output to be in a separate file:

```
PS4='+\011$(date "+%s.%N")\011${LINENO}\011'
exect 5> ${0##*/}.${$.x && BASH_XTRACEFD="5" && set -x
```

Or to get your script looking more professional, one can enable DEBUG, i.e. tracing only happens when you run as DEBUG=profile ./script.sh:

```
case $DEBUG in
  profile|PROFILE|p|P)
```

(continues on next page)
For the larger scripts with loops and functions tracing output with the date stamps and line numbers can be summarized. For further discussion please take a look at 10

Parallel, crontab, perl/awk/sed

Running in parallel with BASH

The shell doesn’t do parallelization in the HPC way (threads, MPI), but can run some simple commands at the same time without interaction.

The simplest way of parallelization is sending processes to a background and waiting in the script till they are done:

```bash
# in the script body one may run several processes, like
command1 &
command2 &
command3 &
```

Here is an example that can be run as `time script` to demonstrate that execution takes 5 seconds, that is the timing of the longest chunk, and all the processes are run in parallel and finished before script’s exit:

```bash
# trap is optional, just to be on the safe side
# at the beginning of the script, to get child processes down on exit
trap 'killall $(jobs -p) 2>/dev/null' EXIT

# dummy sleep commands grouped with echo and sent to the background
for i in 1 3 5; do
    { sleep $i; echo sleep for $i s is over; } &
done

# 'wait' makes sure jobs are done before script is finished
# try to comment it to see the difference
wait
echo THE END
```

Putting `wait` at very end of the script makes it to wait till all the child processes are over and only then exit. Having `trap` at very beginning makes sure we kill all the process whatever happens to the script. Otherwise they may stay running on their own even if the script has exited.

Another way to run in parallel yet avoiding sending to the background is using `parallel`. This utility runs the specified command, passing it a single one of the specified arguments. This is repeated for each argument. Jobs may be run in parallel. The default is to run one job per CPU. If no command is specified before the `--`, the commands after it are instead run in parallel.

```bash
# normally the command is passed the argument at the end of its command line. With -i
# option, any instances of "{}" in the command are replaced with the argument.
parallel command {} -- arguments_list

# example of making a backup with parallel rsync
parallel -i rsync -auvW {}/ user@server:{}.backup -- dir1 dir2 dir3
```

10 https://stackoverflow.com/questions/5014823/how-to-profile-a-bash-shell-script-slow-startup
# in case you want to run a command, say ten times, the arguments can be any dummy
→list
# normally parallel passes arguments at the end of the command, with '-i' they need to be placed explicitly with '()', or can be skipped, like here
parallel -i date -- {1..10}
#
if no command is specified before the --, the commands after it are instead run in parallel,
parallel -- ls df "echo hi"

On Triton we have installed Tollef Fog Heen’s version of parallel from moreutils-parallel CentOS’ RPM. GNU project has its own though, with different syntax, but of exactly the same name, so do not get confused.

**Crontab**

Allows run tasks automatically in the background. Users may set their own crontabs. Once crontab task is set, it will run independently on whether you are logged in to the system or not.

Run `crontab -l` to list all your current cron jobs, `crontab -e` to start editor. When in, you may add one or several lines, then save what you have added and exit the editor normally.

```
# run 'script' daily at 23:30
30 23 * * * $HOME/bin/backup_script > /home/user/log/backup.log 2>&1
# every two hours on Mon-Wed,Fri
0 /2 * * 1-3,5 rm /path/to/my/tmp/dir/* >/dev/null 2>&1
```

The executable script could be a normal command, but crontab’s shell has quite limited functionality, in case of anything more sophisticated than just a single command and a redirection you end up writing a separate script.

The first five positions corresponds to:

- minute (0-59)
- hour (0-23)
- day (1-31)
- month (1-12)
- day of week (0-7, 0 or 7 is Sunday)

Possible values are:

- `*` any value
- `,` value list separator
- `-` range of values
- `/` steps

You set your favorite editor: `export EDITOR=vim` (can be a part of `~/.bashrc`).

As part of the crontab file you may set several environment variables, like `MAILTO=name.surname@aalto.fi` to receive an output from the script or any possible errors. If `MAILTO` is defined but empty (`MAILTO=""`), no mail is sent.
Perl, awk, sed

Powerful onliners. Please consult corresponding man pages and other docs for the details, here we provide some examples. As it was stated at very beginning of the course, shell, with all its functionality is only a glue in between all kind of utilities, like grep, find, etc. Perl, awk and sed are what makes terminal even more powerful. Even though Perl can do everything what can awk and sed, one still may find tons of examples with the later ones. Here we provide some of them.

Python is yet another alternative.

```
# set delimiter to : and prints the first field of each line of passwd file (user name)
awk -F: '{print $1}' /etc/passwd

# sort lines by length, several ways to do it
cat file | perl -e 'print sort { length($a) <=> length($b) } <>'
cat file | awk '{ print length, $0 }' | sort -n -s | cut -d" " -f2-

# placeholders replacement example above could be
NAME=Jussi EMAIL=jussi@gmail.com; sed -e "s/$NAME/$NAME/" -e "s/$EMAIL/$EMAIL/" -e "template"

# inline word replacing in all files at once
perl -i -p -e "s/TKK/Aalto/g" *.html
```

About homework assignments

Available on Triton. See details in the $course_directory.

References

To continue: course development ideas/topics

Additional topics:
- select command
- revise coreutils section, expand the examples and explanations, make it clear that BASH is about getting those small utilities to work together
- benchmark: C-code vs BASH, Python vs BASH, Perl vs BASH

Ideas for exercises

- function to find all broken links
- (homework?) Implement a profiler, that summarizes PS4/date output mentioned above
- (homework?) Script that makes ‘pe1 pe2 … gpu32’ out of ‘pe[1-16],gpu11,gpu32’

In general, there could one script that one starts building from the first line up to a parallelization. Like backup script with rsync.

Git usage?
Aalto scientific computing guide

Bonus material

Parts that did not fit.

[FIXME: should be moved to another tutorial SSH: beyond login]

SSH keys and proxy (bonus section)

- SSH is the standard for connecting to remote computers: it is both powerful and secure.
- It is highly configurable, and doing some configuration will make your life much easier.

SSH keys and proxy jumping makes life way easier. For example, logging on to Triton from your Linux workstation or from kosh/lyta.

For PuTTY (Windows) SSH keys generation, please consult section “Using public keys for SSH authentication” at

On Linux/Mac: generate a key on the client machine

```
ssh-keygen -o # you will be prompted for a location to save the keys, and a
--passphrase for the keys. Make sure passphrase is strong (!)
ssh-copy-id aalto_login@triton.aalto.fi # transfer file to a Triton, or/and any
--other host you want to login to
```

From now on you should be able to login with the SSH key instead of password. When SSH key added to the ssh-agent (once during the login to workstation), one can login automatically, passwordless.

Note that same key can be used on multiple different computers.

SSH proxy is yet another trick to make life easier: allows to jump through a node (in OpenSSH version 7.2 and earlier -J option is not supported yet, here is an old recipe that works on Ubuntu 16.04). By using this, you can directly connect to a system (Triton) through a jump host (kosh): On the client side, add to ~/.ssh/config file (create it if does not exists and make it readable by you only):

```
Host triton triton.aalto.fi
    Hostname triton.aalto.fi
    ProxyCommand ssh YOUR_AALTO_LOGIN@kosh.aalto.fi -W %h:%p
```

Python for Scientific Computing

See also:

This material has been moved to https://aaltoscicomp.github.io/python-for-scicomp/

Abstract

Python is a modern, object-oriented programming language, which has become popular in several areas of software development. This course discusses how Python can be utilized in scientific computing. The course starts by introducing the main Python package for numerical computing, NumPy, and discusses then SciPy toolbox for various scientific computing tasks as well as visualization with the Matplotlib package.

7 https://the.earth.li/~sgtatham/putty/0.70/htmldoc/
Motivation

Why Python

Python has become popular, largely due to good reasons. It’s very easy to get started, there’s lots of educational material, a huge amount of libraries for doing everything imaginable. Particularly in the scientific computing space, there is the Numpy, Scipy, and matplotlib libraries which form the basis of almost everything. Numpy and Scipy are excellent examples of using Python as a glue language, meaning to glue together battle-tested and well performing code and present them with an easy to use interface. Also machine learning and deep learning frameworks have embraced python as the glue language of choice. And finally, Python is open source, meaning that anybody can download and install it on their computer, without having to bother with acquiring a license or such. This makes it easier to distribute your code e.g. to collaborators in different universities.

Why not Python for Scientific Computing

While Python is extremely popular in scientific computing today, there are certainly things better left to other tools.

• Implementing performance-critical kernels. Python is a **very** slow language, which often doesn’t matter if you can offload the heavy lifting to fast compiled code, e.g. by using Numpy array operations. But if what you’re trying to do isn’t **vectorizable** then you’re out of luck. An alternative to Python, albeit much less mature and with a smaller ecosystem, but which provides very fast generated code, is **Julia**.

• Creating libraries that can be called from other languages. In this case you’ll often want to create a library with a C interface, which can then be called from most languages. Suitable languages for this sort of task, depending on what you are doing, could be Rust, C, C++, or Fortran.

• You really like static typing, or functional programming approaches. **Haskell** might be what you’re looking for.

Python 2 vs Python 3

There are two slightly incompatible versions of Python being used today, 2 and 3. Most large projects have supported 3 for a long time already, and have announced dropping Python 2 support for future versions (or have already done so), so at this point you should use version 3 unless you’re working on an existing project that for some reason hasn’t yet been ported to version 3. Accordingly, in this course we will use Python 3. For more info, see the Python 3 statement by many other the major projects.

Practical details

The instructor will use the **anaconda3/latest** module available on triton. However, if you have Python 3 with the usual scientific libraries installed locally on your laptop, you should be able to use that as well, if you prefer.

For interactively testing things in Python, you can use a Jupyter notebook, or the **ipython** shell. For writing Python code you will need a text editor or IDE; Jupyter Lab does have one, if you prefer to work in a browser based environment. Popular free programming text editors or IDE’s with good Python support include:

• Emacs
• Vim
• VS Code
• Spyder
• Eclipse + PyDev
You’re not expected to know much Python at the start of the course, but a little bit of programming proficiency is needed as a prerequisite.

Although not necessary for this course, knowledge of a version control system is useful when programming (or writing papers with LaTeX or other text-based formats). The most common and powerful version control system today is git. To get started with git, see our list of Git tutorials.

The course focuses on hands-on demonstrations and exercises rather than lectures.

Introduction to Python

If you are not familiar with Python, a very short introduction; first, the builtin scalar and collection types:

Scalars

Scalar types, that is, single elements of various types:

```
i = 42    # integer
i = 2**77  # Integers are arbitrary precision
i = 3.14   # floating point number
c = 2 - 3j  # Complex number
b = True   # boolean
s = "Hello!" # String (Unicode)
q = b'Hello' # bytes (8-bit values)
```

Collections

Collections are data structures capable of storing multiple values.

```
l = [1, 2, 3]  # list
l[1]            # lists are indexed by int
l[1] = True    # list elements can be any type
d = {"Janne": 123, "Richard": 456} # dictionary
d["Janne"]
s = set()      # Set of unique values
```

Control structures

Python has the usual control structures, that is conditional statements and loops:

```
x = 2
if x == 3:
    print('x is 3')
elif x == 2:
    print('x is 2')
else:
    print('x is something else')
```

While loops loop until some condition is met:
Aalto scientific computing guide

```python
x = 0
while x < 42:
    print('x is ', x)
    x += 0.2
```

For loops loop over some collection of values:

```python
xs = [1, 2, 3, 4]
for x in xs:
    print(x)
```

Often you want to loop over a sequence of integers, in that case the `range` function is useful:

```python
for x in range(9):
    print(x)
```

Another common need is to iterate over a collection, but at the same time also have an index number. For this there is the `enumerate` function:

```python
xs = [1, 'hello', 'world']
for ii, x in enumerate(xs):
    print(ii, x)
```

### Functions and classes

Python functions are defined by the `def` keyword. They take a number of arguments, and return a number of return values.

```python
def hello(name):
    """Say hello to the person given by the argument""
    print('Hello', name)
    return 'Hello ' + name
```

Classes are defined by the `class` keyword:

```python
class Hello:
    def __init__(self, name):
        self._name = name
    def say(self):
        print('Hello', self._name)
```

### Python type system

Python is strongly and dynamically typed.

Strong here means, roughly, that it’s not possible to circumvent the type system (at least, not easily, and not without invoking undefined behavior).

```python
x = 42
type(x)
x + "hello"
```

Dynamic typing means that types are determined at runtime, and a variable can be redefined to refer to an instance of another type:

9.1. Training 435
Jargon: Types are associated with rvalues, not lvalues. In statically typed language, types are associated with lvalues, and are (typically) reified during compilation.

Organizing Python code

Start Python scripts with

```bash
#!/usr/bin/env python3
```

This ensures you get the correct python3 for the environment you are using.

In general, don’t put executable statements directly into the top level scope in your files (modules), as this code is then run if you try to import the module.

Instead, use this common idiom:

```python
if __name__ == '__main__':
    # your code goes here
```

When developing code it’s often convenient to be able to reload a module into your IPython (or IPython notebook) session without having to restart the entire session. This can be done with the `reload` function:

```python
from importlib import reload
import foo
foo.bar()
# Edit foo.py
reload(foo)
foo.bar()
```

Exercise 1.1

Who needs numpy anyway? Implement matrix multiplication with nested lists as your matrix representation. *Hint for beginners:* Create one function

```python
def creatematrix(n, m):
    # ...
```

which creates an NxM matrix filled with random values (e.g. random.random()). Then create another function

```python
def matrixmult(a, b):
    # ...
```

which multiplies together two matrices a and b.
Exercises 1.2

Let's continue with the previous example, and add some object-oriented scaffolding around our matrix code. Create a Matrix class with a constructor to create the random matrix, and overload the `*` operator to multiply two Matrix instances. Reuse the code from the previous exercise.

Exercise 1.3

The essence of science is experiment and measurement. So let's measure our matrix multiplication implementation, and calculate how fast it can multiply matrices, in terms of “GFlops/s” (Giga floating point operations per second). Hint: A “flop” is a floating point multiply or addition/subtraction. First figure out how many flops are needed to multiply two matrices. Then you need to time it; for this you can use the IPython magic `%timeit` command. And finally, equipped with this information, you can calculate a GFlops/s score for your multiplication method.

Exercise 1.4

Basic file I/O. Run the following python snippet to create a file `pangrams.txt`:

```python
with open('pangrams.txt', 'w') as f:
    f.write("The quick brown fox jumps over the lazy dog
Sphinx of black quartz, judge my vow
The dog ate my homework
Pack my box with five dozen liquor jugs
"")
```

Next, create Python code to read that file, and check each line whether it's a pangram. A pangram is a sentence that uses all the letters of the alphabet.

Enter NumPy

Introduction

The NumPy package provides a N-dimensional array type, and syntax and utility functions for working with these arrays. In contrast to a Python list, a numpy array can only hold elements of the same type. The element type can be seen via the ‘dtype’ attribute.

```python
import numpy as np
a = np.array(((1,2,3),(4,5,6)))
a.dtype
a[0, 0] = "hello"  # error!
a[0, 0] = 2**100    # error!
```

What these restrictions buy you is that the memory layout of a numpy array is very efficient, similar to what you see in low level languages like C or Fortran. This means operating on these arrays is very efficient; in fact, much of the speed advantage of numpy comes from the fact that array syntax is implemented in fast C code.

Due to the memory layout of numpy being compatible with C and Fortran, numpy arrays allow one to use functionality written in these other languages. Much of the SciPy ecosystem (NumPy, SciPy, etc.) consist of python wrappers around widely used and battle-tested numerical libraries written in C or Fortran such as LAPACK and BLAS.

The Python list

9.1. Training 437
a_list = [1, "hello", 1.2]

has roughly the following layout in memory:

In contrast, the NumPy array

```python
n = np.array((1,2,3))
```

has the memory layout like

Exercise 2.1

1. In the example above we saw that $2^{100}$ was too large. What is the default datatype of a numpy integer array if we don’t explicitly specify some type, and what is the largest possible integer we can store in such an element.

2. What is the smallest negative element (that is, the largest absolute value of a negative number)? Is it different from the largest positive number, and if so, why?

3. What is the absolute value of the smallest negative element? Why?

Other ways of creating NumPy arrays

There are many different ways to create NumPy arrays, here’s a few of the most common ones:

```python
np.zeros((2, 3))  # 2x3 array with all elements 0
np.ones((3, 2), bool)  # 3x2 boolean array
np.arange(3)  # Evenly spaced values in an interval
np.linspace(..)  # similar to above
```

NumPy array slicing syntax

NumPy provides a convenient array syntax to reference subarrays, similar to MATLAB for Fortran.

```python
a[low:high:step]
```

returns the array elements in the range $[\text{low}, \text{high})$ with a stride of $\text{step}$. Equivalently for multidimensional arrays. For multidimensional arrays NumPy by default stores arrays in row-major order, like C. Note that this is in contrast to e.g. Fortran, MATLAB or Julia that use a column-major layout.

Using array syntax efficiently is key to using NumPy in a fashion that leads to short as well as efficient code.

NumPy also provides so-called advanced indexing, where you can select elements with a list of indices.

```python
a = np.zeros((3, 3))
b = a[(0, 1), (1, 1)]
b[0] = 1  # Will this modify a?
```
Views vs. copies

When slicing an array, you **DO NOT** get a copy of those elements, but rather a view. That is, the data elements are the same as in the original array:

```python
a = np.ones((2, 2))
b = a[1, 1:2]
b[0] = 2
```

Views rather than copies is more efficient, particularly for large arrays, but they can sometimes be confusing. Be careful!

If you do need a copy, NumPy arrays have a `copy` method to create a copy rather than getting a view.

**NOTE** With advanced indexing, you always get a copy!

Array shape and size

NumPy arrays have a shape and size attribute:

```python
a = np.zeros((2, 3))
a.size  # Number of elements
a.shape  # shape tuple
```

We can modify the shape of an array with the `reshape` or `resize` methods. Or for the special case of flattening an array to a 1D array, `ravel`.

Combining, splitting and rolling arrays

For combining multiple arrays into a larger array, see the `concatenate`, `stack`, `block`, and the more specialized variants `hstack`, `vstack`, `dstack`.

Similarly, for splitting an array into multiple parts, there’s `split`, `hsplit`, `vsplit`.

To roll an array, that is shift the elements along a given axis, use `roll`.

Exercise 2.2

Create an array `x` of 100 evenly spaced numbers in the range [-2*pi, 2*pi].

Next, create an array `y`, where each element is the `sin` of each element in the previously created array.

Then, figure out the indices where the array `y` changes sign. What are the `x` values for these indices?

NumPy I/O

NumPy has functionality for saving and loading NumPy arrays from files. For reading/writing textfiles there is `loadtxt` and `savetxt`. See also `genfromtxt` with more sophisticated handling of missing values etc.

For large arrays, it’s faster to use a binary format. For these NumPy defines a `.npy` format. Loading and saving these files can be done with the `load` and `save` methods. There’s also the `.npz` format, which is a zip archive containing several numpy ndarrays in one file. `.npz` format files can be read/written with `load`, `savez` and `savez_compressed` methods. This is a good choice for temporary or intermediate files such as checkpoints.
etc. Note that the format is Numpy-specific, and other languages might not easily be able to read it. Similarly, for long-term archiving other formats might be a better choice.

Random Numbers in NumPy

The `numpy.random` module contains functionality to create pseudorandom numbers following different distributions.

Linear algebra in Numpy

The `dot` method provides a generalized dot product. It can compute dot products of 1D vectors, matrix-vector products as well as matrix-matrix products. It is an interface to the famous BLAS library, of which multiple highly optimized versions exist. The `numpy.linalg` module contains interfaces to the most common linear algebra operations, such as calculating eigenvalues, Cholesky and singular value decompositions, solving linear systems, least squares, (pseudo)inverse. This module is an interface to the LAPACK library (which in turn builds on top of BLAS).

Exercise 2.3

Remember our first exercise, implementing matrix multiplication? Now do the same, but use NumPy arrays and the `dot` method. Compare performance to the code you wrote yourself earlier, using the IPython `%timeit` macro.

Exercise 2.4

Here’s a number of quick numpy exercises to get you a feel of numpy functionality, index manipulation etc.

1. Reverse a vector. Given a vector, reverse it such that the last element becomes the first, e.g. [1, 2, 3] => [3, 2, 1]
2. Create an identity matrix of size 4x4.
3. Create a 2D array with zeros on the borders and 1 inside.
4. Create a random array with elements [0, 1), then add 10 to all elements in the range [0.2, 0.7).
5. What is `np.round(0.5)`? What is `np.round(1.5)`? Why?
6. In addition to `np.round`, explore `np.ceil`, `np.floor`, `np.trunc`. In particular, take note of how they behave with negative numbers.
7. Recall the identity $\sin^2(x) + \cos^2(x) = 1$. Create a random 4x4 array with values in the range [0, 10). Now test the equality with `np.equal`. What result do you get with `np.allclose` instead of `np.equal`?
8. Create a 1D array with 10 random elements. Sort it.
9. What’s the difference between `np_array.sort()` and `np.sort(np_array)`?
10. For the random array in question 8, instead of sorting it, perform an indirect sort. That is, return the list of indices which would index the array in sorted order.
11. Create a 4x4 array of zeros, and another 4x4 array of ones. Next combine them into a single 8x4 array with the content of the zeros array on top and the ones on the bottom. Finally, do the same, but create a 4x8 array with the zeros on the left and the ones on the right.

More quick NumPy exercises like this one over here.
Exercise 2.5

The topic of this exercise is `np.einsum` which implements the ‘Einstein summation convention’. The Einstein summation convention is a commonly used when working with tensors, but can also be useful for succinctly representing array expressions.

For instance, matrix multiplication can be expressed as

$$ C_k^i = A_{ij}^i B_{jk}^j $$

Implement this with the help of `np.einsum`.

Einstein notation is also available in pytorch and tensorflow. For more information about Einstein notation in NumPy see

- A basic introduction to NumPy’s einsum
- Einsten Summation in NumPy
- Einsum is all you need - Einstein summation in deep learning

SciPy

SciPy is a library that builds on top of NumPy. It contains a lot of interfaces to battle-tested numerical routines written in Fortran or C, as well as python implementations of many common algorithms. Briefly, it contains functionality for

- Special functions (Bessel, Gamma, etc.)
- Numerical integration
- Optimization
- Interpolation
- Fast Fourier Transform (FFT)
- Linear algebra (more complete than in NumPy)
- Sparse matrices
- Statistics
- More I/O routine, e.g. Matrix Market format for sparse matrices, MATLAB files (.mat), etc.

Exercise 3.1

Using scipy, calculate the integral of the function $\sin$ in the interval $[0, \pi]$, and compare with the analytical result.

Exercise 3.2

Use the SciPy sparse matrix functionality to create a random sparse matrix with a probability of non-zero elements of 0.05 and size 10000 x 10000. The use the SciPy sparse linear algebra support to calculate the matrix-vector product of the sparse matrix you just created and a random vector. Use the `%timeit` macro to measure how long it takes. Does the optional `format` argument when you create the sparse matrix make a difference?

Then, compare to how long it takes if you’d instead first convert the sparse matrix to a normal NumPy dense array, and use the NumPy `dot` method to calculate the matrix-vector product.
Can you figure out a quick rule of thumb when it’s worth using a sparse matrix representation vs. a dense representation?

**Matplotlib**

Matplotlib is the ‘standard’ Python plotting library. It is quite full-featured, and provides a MATLAB-like plotting API.

To use it, typically you start with

```python
import matplotlib.pyplot as plt

# ...
x = linspace(-4, 4)
plt.plot(x, np.sin(x))
plt.show()
```

When using Jupyter notebooks, use the magic

```
%matplotlib inline
```

which will cause matplotlib plots to appear inline in the notebooks. Very convenient for quick analysis!

Matplotlib has two slightly different interfaces, a state machine interface similar to MATLAB and an object based interface. The state machine interface is quick and easy to get started, but since it’s based on hidden global state behind the scenes, for more complex stuff it might get confusing. Below is an example using the state machine interface.

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0.0, 3.0)

y1 = np.cos(2 * np.pi * x) * np.exp(-x)
y2 = np.cos(2 * np.pi * x)

plt.subplot(2, 1, 1)
plt.plot(x, y1, 'o-')
plt.title('A tale of 2 subplots')
plt.ylabel('Damped oscillation')

plt.subplot(2, 1, 2)
plt.plot(x, y2, '.-')
plt.xlabel('time (s)')
plt.ylabel('Undamped')

plt.show()
```

And here is the same thing, but using the object-based interface

```python
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0.0, 3.0)

y1 = np.cos(2 * np.pi * x) * np.exp(-x)
y2 = np.cos(2 * np.pi * x)
```

(continues on next page)
fig = plt.figure()
ax = fig.add_subplot(211)
ax.plot(x, y1, 'o-')
ax.set_title('A tale of 2 subplots, OO style')
ax.set_ylabel('Damped oscillation')

ax = fig.add_subplot(212)
ax.plot(x, y2, '.-')
ax.set_xlabel('time (s)')
ax.set_ylabel('Undamped')
plt.show()

Exercise 4.1

Try to recreate the figure below:

Exercise 4.2

Create 1000 normally distributed numbers with $\mu = 0$ and $\sigma = 10$. Then create a histogram plot with 50 bins.

Exercise 4.3

Often it’s useful to be able to plot things on a logarithmic scale. Create a plot with 4 subplots, one with a linear scale, one with logarithmic scale on the x-axis, one with logarithmic scale on the y-axis. Then create 4 functions, such that each will produce a straight line in one of the plots, and plot them.

Image and pseudocolor plots

Matplotlib can also plot 2D data such as images. A common type of 2D plot is the pseudocolor plot, where you want to convert a scalar value in some range into a color value. This means that you must map the range of values into a colormap. Things to think about when selecting a colormap:

- The map should be perceptually uniform. From https://bids.github.io/colormap/ : A “perceptually uniform” colormap is one for which the “perceptual deltas” plot makes a simple horizontal line. (This is essentially the derivative of the colormap in perceptual space with respect to the data. We want our colormap to have the property that if your data goes from 0.1 to 0.2, this should create about the same perceptual change as if your data goes from 0.8 to 0.9. For color geeks: we’re using CAM02-UCS as our model of perceptual distance.)

- It should look good when rendered in gray-scale, e.g. if someone prints the picture on a black-and-white printer. Or sometimes per-page costs in journals are lowers if everything is BW.

- It should make sense to people with the most common type of color blindness (red-green). In practice this means the color maps shouldn’t use both green and red colors, so that they are not confused.

**Bad news**: The commonly used rainbow (or “jet” as it’s often called) is very bad when comparing against these criteria! NEVER USE IT! It was the default in matplotlib < 2.0, and in MATLAB for a long time.
**Good news:** Matplotlib >= 2.0 has sane defaults here! See changes in default styles for matplotlib 2.0. In particular, compare matplotlib < 2.0 default colormap and the matplotlib >= 2.0 default colormap. As an aside, the 2.0 default colormap (“viridis”) is similar to the default colormap in current versions of MATLAB (“parula”).

One case where you should NOT use viridis is if your data has some “natural” zero point. In that case it’s better to use a “cool-warm” style colormaps, see “Diverging Colormaps” at the matplotlib colormap reference.

```python
N = M = 200
X, Y = np.ogrid[0:20:N*1j, 0:20:M*1j]
data = np.sin(np.pi * X*2 / 20) * np.cos(np.pi * Y*2 / 20)

fig, (ax2, ax1) = plt.subplots(1, 2, figsize=(7, 3))
im = ax1.imshow(data, extent=[0, 200, 0, 200])
ax1.set_title("v2.0: 'viridis'")
fig.colorbar(im, ax=ax1, shrink=0.8)

im2 = ax2.imshow(data, extent=[0, 200, 0, 200], cmap='jet')
fig.colorbar(im2, ax=ax2, shrink=0.8)
ax2.set_title("classic: 'jet'")
fig.tight_layout()
```

**Exercise 4.4**

Find an image on the internet or already on the machine you’re working on, load it into a NumPy array (you can use imageio.imread() for this), and see if you can create an Andy Warhol-like print by using a suitable colormap.

**Demo application**

To demonstrate how to make a simple simulation program, here the lecturer will ‘live-code’ a small simulation program. For a suitable model, lets choose a suitably fascinating problem that can be simulated with a relatively simple model. First, some background.

**Topological phase transitions**

Historically, for a long time we believed there were two, and only two, kinds of phase transitions in nature. So-called discontinuous, or first-order, transitions which are characterized by the presence of a latent heat (mathematically, a discontinuity in the first derivative of the free energy with respect to some thermodynamic parameter), whereas continuous phase transitions are characterized by a discontinuity in the second or higher derivative of the free energy.

However, in the 1970’ies, some experiments on ultrathin films of superfluid Helium-3 were made which produced data that existing theories could not describe. Eventually Kosterlitz and Thouless (and independently Berezinskii in the then Soviet Union) were able to describe what was happening. What they had discovered was an entirely new kind of phase transition which defied the existing classification schemes. Namely, there is **NO** discontinuity in any free energy derivative. So in a way, it’s an infinite-order phase transition.

What is happening is that topological defects (vortices in this case) in the system change how they interact with each other at the critical temperature. At low temperatures below the transition temperature the correlation function between spins decays as a power law, whereas above the transition temperature the correlation decays exponentially. This results in vortex-antivortex pairs at low temperature, and a vortex unbinding transition at the transition temperature with free vortices at higher temperatures.

This work eventually resulted in the 2016 Nobel Prize in Physics. See the scientific background for the 2016 physics prize.
The XY model

Topological phase transitions can be studied with a XY model (also called the planar model, or rotor model). Take a lattice with spins rotating in the plane. Each spin interacts with its neighbors, and the configuration energy of the system is given by

\[ E = -J \sum_{i \neq j} s_i s_j, \]

where the sum is over nearest neighbor spins.

In this case we can ignore the constant J which determines the interaction strength. Also, since the spin vectors are all of equal lengths the dot product can be simplified, so we have

\[ E = -\sum_{i \neq j} \cos(\theta_i - \theta_j). \]

The Metropolis-Hastings Monte Carlo algorithm

The Metropolis-Hastings algorithm is a Markov chain Monte Carlo method that can be used for sampling a probability distribution. In this case, the basic idea is that for each spin \( s \) we do a trial move, to change the spin. We then calculate a random trial spin \( s' \), and calculate an acceptance probability

\[ A = \min(1, \frac{P(s')}{P(s)}). \]

In this case the probability density is the Boltzmann distribution

\[ P(s) = \frac{1}{Z} \exp(-\beta E(s)), \]

where \( \beta \) is the thermodynamic beta, or

\[ \beta = \frac{1}{k_B T}, \]

where \( k_B \) is the Boltzmann constant. For this simulation we can set it to 1 and ignore it hereafter. \( \beta \) is thus just the inverse of the temperature.

Thus the quotient

\[ \frac{P(s')}{P(s)} \]

can be calculated as

\[ \exp(-\beta(E' - E)). \]

Then finally, calculate a uniform random number \( r \) in the interval \([0, 1)\). If \( r \leq A \) the new state is accepted. Repeating this for all the spins constitutes a single Monte Carlo step in the algorithm.

Other useful Scientific Python libraries

A few other widely used libraries in the Scientific Python ecosystem:

- **Pandas**: Python Data Analysis library. Pandas gives Python a dataframe type, similar to data frames in R, which is useful for representing tabular data where every column can be of a different type. If you’re interested in this topic, see the *Practical R and Python Data Analysis* course by Aalto Science-IT.
Homework: 2D Ising model

This homework exercise shares many similarities with the XY model studied above. The main difference is that in the 2D Ising model, the spins are perpendicular to the plane, and can take only two values, $+1$ and $-1$. This model can be used to study the ferromagnetic phase transition. Below the critical temperature ferromagnetic domains, where the spins are aligned, form. Above the critical temperature this order breaks down. In the Ising model the configuration energy is defined as

\[ E = -J \sum_{i \neq j} \sigma_i \sigma_j - \mu H \sum_j \sigma_j, \]

where $J$ is the exchange energy, $\mu$ is the magnetic moment of the spins, and $H$ is the external magnetic field in the direction perpendicular to the plane. To simplify, you can set $J$ and $\mu$ to 1.

Implement a simulation program simulating the 2D Ising model. Use the Metropolis-Hastings Monte Carlo algorithm. Visualize the results with matplotlib. Run the simulation at different temperatures and with different starting configurations (random vs. ordered), and see if you can find the critical temperature by observing your visualizations.

If you find the above too easy, a few topics for further exploration. Not needed to pass the course.

- Implement the Wolff algorithm, which flips whole clusters at a time instead of individual spins. This helps avoid a phenomena called critical slowing down close to the critical temperature, which is problematic for algorithms such as the Metropolis algorithm that flip one spin at a time.
- Calculate and plot the net magnetization, the magnetic susceptibility, and the heat capacity of the system as a function of the temperature. How do they behave around the critical temperature?

9.1.5 Recommended programming courses

Need to learn programming? We will include some recommended online programming courses here.
10.1 Help

There are many ways to get help with your scientific computing and data needs - in fact, so many you don’t know what to use. This page lists how to ask for help, for different kinds of needs.

10.1.1 Formulate your question

We get many requests for help which are too vague to give a useful response, so we delay while we try to find something better than “please explain more”, which slows everything down. So, when sending us a question, always try to clarify these points to get the fastest solution:

- **Has it ever worked?** (If so, what has changed?)
- **What are you trying to accomplish?** (Your ultimate goal, not current technical obstacle.)
- **What did you do?** (Be specific enough to be reproducible - copy and paste exact commands you run, exact output messages, scripts, inputs, etc.)
- **What do you need?** Do you need a complete solution, pointers to get started, or should we say if it will take too long and we recommend you think of other solutions first?

If you don’t know something, it’s OK, just explain the best you can and we’ll go from there! You can also chat with us to brainstorm about issues in general, which helps to figure out these questions. A much more detailed guide is available from Sigma2 documentation.

We don’t need a long story in the first message - we’ll ask for more later. Try to cover these points, and we are happy to get your message.

10.1.2 Aalto Scientific Computing

Aalto Scientific Computing (Science-IT) is focused on all aspects of computing and data, and mostly consist of PhD-level researchers so we can understand what you are doing, too. Our main focus areas are high-performance computing *(Triton)*, research software *(RSEs)*, data, and training *(training)*.

- Problems with Triton, using Triton
- Help with software on Triton
- Data advice, FAIR data, confidential data, data organization
- Suggestions on tools and workflows to use
- General research software and research tools
Aalto scientific computing guide

- Advice on other Aalto services
- Advice on using CSC services
- Triton Accounts (by email)
- Increasing quotas, requesting group storage space (by email)

Scicomp garage

Summer News!
During the summer we have reduced support, but you are not alone.
- Please note that there will be no garage on 3, 7, 8, 9 June. (Teaching)
- From 21/June until 8/August the garage is best-effort: we might be there, but probably not every day in the high summer.
- During summer, try to ask in advance, either the issue tracker or chat and we’ll tell you when we can meet you.
- For quick things (e.g. “is it just me or is everything down?”) join our chat http://scicomp.zulip.cs.aalto.fi/

Check back here for updates about our actual availability.

If you need more help than the issue truckers, this is the place to be. It’s not just Triton, but all aspects of scientific computing.

Come if you want to:
- Solve problems
- Discuss and figure out what your problem really is
- Brainstorm the best strategy are for your problems
- Work with someone on your issues in real time
- Network with others who are doing similar work and learn something new

Notes:
- All garages are open to all people (but see the specialities).
- You don’t have to have a specific question, you can come by just to chat, listen, or figure out if you should have a question.

Online

Triton, SciComp, RSE, and CS

Link
https://aalto.zoom.us/j/65371910202, every day at 13:00 (NOTE: not guaranteed from 21/June until 8/August), check this page for info.
You can meet us online, every weekday, at 13:00, online via zoom. Imagine this like walking into our office to ask for help. If you are not sure whether we can help you, come and chat with us anyway and we can figure out a solution together.

- This doesn’t replace email or the Triton issue tracker for clearly-defined tasks. Garage is good for discussion, brainstorming, and deciding the best path. If in doubt, come to garage and we will help you decide. Many people make an issue, then come to garage to discuss.

- Arrive between 13:00 - 13:15. We leave at 13:15 if there is no one around (and the person you need to talk to may leave). Please don’t arrive early since we another meeting then. Program always ends at 14:00.

- We have some focus days (see list below) to ask about specific topics and make sure you find the right person to help you, but you can always visit on any day and ask anything not related to the “focus”.

- Join on Zoom via https://aalto.zoom.us/j/65371910202. If joining by web browser, use Chrome. Audio doesn’t work with Firefox or Safari.

Focus days

We have added some focus days (see list below), but you can always visit on any day and ask anything not related to the “focus”.

- **Mondays** no special focus, come ask anything.

- **Tuesdays** are Research Software Engineering focus days: Come ask about issues with your code or software tools you use. We can also help with basic methodological or statistical issues.

- **Wednesdays** are Data focus days, this is in joint collaboration with the Aalto Data Agents network from all Aalto schools. Come ask about research data management, data sharing/reusing/opening, data versioning, sensitive data, data management plans, legal and ethical issues with research data, workflows for working with big datasets.

- **Thursdays** are Triton focus days: Come ask about code parallelization, slurm job submissions, and usage of Triton cluster.

- **Fridays** are CS focus days.

NBE/PHYS Online

This garage combines NBE, PHYS, and ITS (Aalto IT Services) staff, both from the computational and non-computational sides.

This garage happens on Zoom every Monday/Wednesday/Friday from 11:00 to 12:00. The link is sent to department personnel lists.

In person

General garage (CS Building)

In-person garages suspended for the time being for the obvious reason. The online garage above is more frequent and even better.
NBE garage, F-building

NBE garage used to happen in person every first wednesday of the month, room F336 13:00. NBE garage covers also issues related to working with personal data, research ethics, best practices in biomedical research.

Due to the current situation NBE garage happens online at the same time of the Aalto Sci Comp garage (see above).

Past events

Scicomp Garage has existed since Spring 2017.

SciComp community

Let’s face it: we learn more from each other than from classes. There is a major problem with inequality in computational sciences, a large part is related to how we learn these tools. Join the Aalto Scientific Computing community to help you and others be the best scientist you can be. You can

- Network with others within a supportive mentoring community.
- Share knowledge among ourselves, avoid wasting time on things where someone knows the answer.
- Take part in developing our services - basically, be a voice of the users.

SciComp Garage and issues

Currently, most of our interaction happens in the daily SciComp Garage, which is a daily meeting where we help others (and learn ourselves). If you hang out there, you will learn a lot.

If you subscribe to the Triton issue tracker, you will see a lot of questions and answers, and thus learn a lot.

Aalto community chats

We have weekly chats for the Aalto scientific computing poweruser/RSEs as a way to network with the community and Aalto staff. Currently, these are done at 10:00 on Thursdays as part of the Nordic-RSE Finland chats. Anyone is welcome to join and discuss Aalto-related topics.

Mailing lists

- If you have a Triton account, you are on the triton-users mailing list already. Many training and other events are announced there.
- If you do not have a Triton account, the scicomp-announcements mailing list provides the same information. Subscribe here.
- Join our Research Software Engineer mailing list for information on research software related topics and the RSE community at Aalto.
Chat

- Join the Aalto Scientific Computing group on Aalto Microsoft Teams. The invite code is e50tyi j. In practice, we watch it for questions but it’s not the most active place (but it could be).
- We often hang out on the CodeRefinery chat, and there is an #aalto stream there.

External links / Social media

- Github: AaltoSciComp
- Twitter: @SciCompAalto
- YouTube: Aalto Scientific Computing

User groups

Often, there is specialized software or problem domains which need more advanced documentation than the generic HPC talks. Often, the SciComp staff aren’t experts in this particular domain, so we can’t provide immediate help without knowing more. For this, we have user groups: we meet with groups of users to discuss problems and create solutions/documentation about them.

Existing user groups

To be formed.

If you would like to create a user group, let us know. The hardest part is finding the users, so if you form the group of people and schedule a time, it is very easy for us to come. **To be clear, if you bring people together and want to organize the group, we are very happy and will take part and make it “official”.**

User group meetings

A user group meets periodically, and does various things. At the meeting is some SciComp staff as well as interested users, who want to make a larger change than just solving their own problems.

- See examples of the software or problem in practice.
- Discuss the best solution of problems
- Collaboratively create documentation on the problem (which can be put straight at scicomp.aalto.fi, for example in **Applications: General info**). We can create video demos, **examples**, and more.
- Discuss how the infrastructure needs to be adapted to the actual use cases.
- Provide a network for informal support within research groups.
Website

Search this website for help. For that matter, also search the internet in usual. This is usually a good place to start, but often you need to move on to the next steps.

Issue tracker

The Triton issue tracker, which is where most Triton and scientific computing issues should go, even if not directly Triton related. *Log in and search the issue tracker for related issues, you may find the solution already*

If you issue is about Triton, software, or somewhat related, this is where it should go.

Garage

Link

[https://aalto.zoom.us/j/65371910202](https://aalto.zoom.us/j/65371910202), every day at 13:00

Daily *SciComp Garage sessions*, where you can informally chat. This is especially useful when your question is not yet fully defined, or you think that demonstrating the problem for immediate feedback is useful.

Chat

Chat can be a great way to quickly talk with others, share tips, and quickly get feedback on if a problem is large or small, something to get help with or figure out yourself, etc. For longer solutions, we will direct you to the issue trackers but it rarely hurts to do a real-time discussion. *(For real-time video chat with screen sharing, come to the garage above).*

The [SciComp Zulipchat](https://scicomp.zulip.cs.aalto.fi) is where we most often hang out. You can ask triton questions in #triton, general questions in #general, research software engineering questions in #rse, etc. The main point of Zulip is **topics**, which allow you to name threads and easily follow old information. *(use zulip in your courses)*

You can also chat with us on Aalto Microsoft Teams. The invite code is e50tyij. We are also findable on various other department chats here.

Email

- scicomp at aalto.fi. Use this only for things related to your account (requesting a Triton account), quota, etc. - most other things go to the tracker above.
- rse-group at aalto.fi: Research software engineering service requests.
10.1.3 Department IT

CS, NBE, and PHYS have their own IT groups (among others, but those are the Science-IT departments with the most support). They handle local matters and can reliably direct you to the right resources. Department IT handles:

- Computers, laptops, personal devices
- Department data storage spaces
- Other department-managed tools and services

Reach them by department-specific email addresses

NBE and PHYS IT use the same email issue tracker (esupport) as Aalto IT, so issues can be exchanged no matter which address you send an issue to. CS uses a different one, so you have to think a bit more before sending something.

10.1.4 Community

In addition to formal support, there is informal activities, too:

- The daily SciComp Garage, designed to provide one-on-one help, but we invite anyone to come, hang out in the main room, and network with us. This is for basic help and brainstorming.
- Subscribe to notifications from the Triton issue tracker even if you don’t post there. You will learn a lot.
- Sign up for the Research software engineers and powerusers mailing list and learn about more events that interest you. This isn’t the place to ask for basic help, but if you hang out here you will learn a lot.

10.1.5 Other groups at Aalto

servicedesk, Aalto IT

servicedesk@aalto.fi is the general IT Services service desk. They can handle things with account, devices, and so on. They have a wide range of responsibilities, but don’t always know about local resources that may be more appropriate for your needs. There is an “IT Services for Research” group which focuses on research needs

For students (who aren’t also researchers), this is always your first point of contact - in addition to your teacher.

servicedesk handles:

- Aalto accounts, passwords (including Triton passwords)
- University-wide data storage (work, teamwork, home directories)
- All university-wide common IT infrastructure: wifi, network, devices, websites, learning platforms, etc.
- Anything department stuff, when you are not in a department with local IT staff.

Reach them by:

- Email or phone
- Browse the IT Services for Research list
Research services

Aalto Research Services function more as project administrative services rather than close research support. However, they provide important information for:

- Data management plans for funding applications
- Legal or ethical advice, making contracts and NDAs.
- Library services
- Applying for funding and administering it.

In many cases, you can chat with Aalto Scientific Computing and we can give some initial practical advice.

Reach research services by:

- Contacting service email addresses at the link above
- Contacting school representatives findable at the link above

10.2 About

Computational research is one of the focus areas in Aalto University, and Aalto Scientific Computing makes that possible.

The Science-IT project was founded in 2009 (with roots going back much further) and has since expanded from high-performance computing services to a complete package: we provide computation, data management, software, and training. Our partnerships with departments and central IT services allow a streamlined experience from personal devices to the largest clusters.

To reflect our expanded services, we have rebranded to Aalto Scientific Computing to reflect our greater mission and partners.

Many Centres of Excellence and departments at Aalto University are using our resources with great success. There are currently over 1000 user accounts from all six different schools and at least 14 different departments using our resources. Science-IT is administered from the School of Science with additional university-level funding - our HPC services are available to all Aalto University, free of charge.

10.2.1 Boilerplate text for grant proposals

Below are various texts which describe our resources, suitable for inclusion in grant applications an the like. There are various types suitable for different purposes.

If you create your own texts and would like to share them, send them to us.

Focus on Triton

Computing and modelling are strategic areas of Aalto University. To support research in these scopes the university is committed to provide proper hardware resources and supporting personnel on long term basis. Currently we provide a system with about 10000 computing cores. The System also contains 150 NVIDIA cards for GPU computing and over 5 PB of fast storage capacity suitable for Big Data needs. All parts are connected with a fast Infiniband network to support parallel computing and fast data access. To keep the resources competitive we annually upgrade the system based on the needs of researchers.

All resources are integrated with the national resources allowing easy migration to even larger resources when necessary. These include e.g. University dedicated OpenStack based cloud resources and access to thousands of servers via
the national computing grid. Furthermore we provide much preconfigured software and hands on support to make the usage for researchers as effective as possible. On the personnel side we have five permanent Ph.D. level staff to keep the system running and providing teaching and consultation for researchers.

Focus on data

Computing and data are strategic areas in Aalto University.

The university provides data management and computing solutions throughout the data lifecycle. The university provides free storage to researchers of essentially unlimited size, provided that the data is managed well. Data storage includes 5PB of high-performance Lustre filesystem space connected directly to a computing cluster for efficient and secure analysis, and 1PB of reliable, backed-up storage space for longer-term storage. Expert staff, both technical and administrative, provide advice and hands-on support in data storage, computation, FAIR principles, data management planning, as well as computation.

Data management is designed with a focus on security. Recommended storage locations are centrally located for security. Computing nodes and data storage servers are physically located at CSC, Keilaranta 14, Espoo. The server room is certified security level 3 (VAHTI-3) i.e. only authorized personnel with clearance are given access to it and there is continuous camera surveillance. All data is access controlled by passwords and individual-level authorization, and firewalled to university networks.

Research environment: research software engineers

The Aalto Research Software Engineer (RSE) team provides a specialized advice and service in research software, data, and computing so that any researcher can accomplish the best science without being held back by technological problems. Typical tasks including implementing a method better or faster than could otherwise be done, or ensuring that results are as open and reusable as possible so that the full impact of the work can be realized. RSE staff are professional researchers with years of experience in computational sciences, and work seamlessly with the rest of the Science-IT team. For the School of Science, basic services is included as part of overheads, or longer-term services can be funded from specific research projects.

Research software engineering services

(this text must especially be tuned to your grant, replace the parts in CAPITAL LETTERS)

This grant will make use of the Aalto Research Software Engineer program to hire high-quality TOPIC specialists. This program provides PhD-level personnel to work on THINGS, which allows the other staff on this project to focus on YYY. Research software engineers do not need to be independently recruited, and are available for consultation also before and after the project. This service is provided by Aalto Scientific Computing, which also provides high-performance computing resources for our project. The Research Software Engineering service is integrated into computing services as a consistent package.

(for basic service, for now only SCI) The service is available as a basic consulting service for free.

(for paid services) This project receives dedicated service from the Research Software Engineering group, funded as researcher salary from this grant. During this period, one of the Aalto research software engineers joins this project as a researcher, equal to all other project employees.
Acknowledgement for papers

Remember you need to in your papers if you use it. See the acknowledging Triton page for instructions on how to do that and some boilerplate text

10.2.2 Usage model and joining

Aalto Scientific Computing operates with a community stakeholder model and is administered by the School of Science. Schools, departments, and other units join and contribute resources to get a fair-share of the output. There are two different components to join:

- **HPC: Science-IT.** Get a share of computing resources via the Triton computing cluster.
- **Aalto Research Software Engineers (RSE):** Support of the RSE program provides intensive hands-on support and service for research software development.

For everyone

Aalto Scientific Computing gets university-level support already, so our computing resources are usable by anyone doing research at Aalto (with a limited share). By joining further, a unit gets something even more valuable: time. Our support for using our infrastructure is concentrated for member departments which provide joint staff with us or support the RSE program, in addition to a greater share of resources.

Staff network

*There is no Aalto Scientific Computing, just people who want to make computing better.*

You might be a department IT staff member, a lab engineer, a skilled postdoc or a doctoral candidates who helps other researchers with their technical/computational challenges. Why not joining forces and join our network of specialists? There is no “Aalto Scientific Computing” on paper, only different teams that work together to help researchers better than they could alone. We invite interested staff to join our community, help sessions, infrastructure development, etc. This program is just being developed (as of 2020), but it roughly includes:

- Participation in admin meetings to help us develop infrastructure (e.g. Triton) in the best way for your users
- **Teaching,** for example ensure our classes are suitable to your audience, teach your own classes with our help via CodeRefinery, or directly help us teach.
- Co-maintenance of infrastructure (for example, your unit’s special software) on Triton and in our automated software deployment systems.
- Learn how to solve your users’ problems more efficiently.
- Networking and continual professional development
- This is not just for IT support or administrative support, but high-quality research support that connects all aspects of modern work.

This does not replace local support, it just makes it more powerful.

Todo: How to take part.
Triton: computing and data storage resources

*Triton* is the Aalto computing cluster, for computationally and data-intensive research. Users from members of the community are allocated resources using a fair-share algorithm that guarantees a level of resources at least proportional to the stake, without the need for individual users to engage in separate application processes and billing.

Each participating department/unit funds a fraction of costs and is given an agreed share of resources. These discussions are carried out with the board of the Science-IT project. Based on this agreed share, units cover the running expenses of the project. There is also direct Aalto funding, which allows the entire Aalto community to access a share of Triton for free.

However, computing is not just hardware: support and training is just as critical. To provide support, each unit that is a full member of Science-IT is required to nominate a local support contact as their first contact point. Our staff tries to provide scientific computing support to units without a support contact on a best-effort basis (currently, that effort is good), but we must assume a basic level of knowledge and attendance at our training courses.

Interested parties may open discussion with Science-IT at any time. Using our standing procurement contracts, parties may order hardware to be integrated into our cluster with dedicated or priority access (or standalone usage), allowing you to take advantage of our extensive software stack and management expertise, with varying levels of dedicated access: a share of total compute time, partitions with priority access, private interactive nodes, and so on. Please contact us for details.

Scientific software: research software engineers

The *Research Software Engineer program* provides specialists in software and data, who can be contracted out to projects to provide close support. The goal is not just to perform a service, but to teach by hands-on mentoring.

For projects, the principle is that the project pays for help lasting more than a few hours or days. This can seamlessly come from project money as a researcher salary.

Units (departments, schools) can also join to get a basic service - their members can receive short-term support without any billing needed. Their members will also receive priority for the project services.

For more information, see the *RSE for units* page.

Contact

Let Mikko Hakala know about Science-IT related joining, Richard Darst know about the RSE program or SciComp community, or contact us at our scicomp.aalto.fi email address.

10.2.3 What we do

We don’t just provide computing hardware, but a complete package of infrastructure, training, and hands-on support. All of these three activities feed back into each other to improve the whole ecosystem.
We provide many types of services:

- Plentiful data storage
- Integrated to compute and other systems
- High-level management in addition to raw storage
- Motivates good management practices

- Relieve users from burden of installation
- Deploy software effortlessly and automatically
- Help with users' own software

- Practical skills not learned academically
- For all levels and backgrounds

- HPC
- Devices
- GPUs
- Cloud

Aalto scientific computing guide
10.2.4 Our components, partners, and collaborators

Aalto Scientific Computing serves as a hub of computational science at Aalto. We guide researchers to the right service, regardless of who is providing it.

Science-IT serves as the coordinator, and runs the Triton cluster, the physical hub of large scale computational and data-intensive research at Aalto. As such, we maintain many active collaborations which allow us to guide researchers to the right resource, regardless of who provides it.

Science-IT

Science-IT (Aalto HPC)

Science-IT is the formal name of the project which provides the Triton computational cluster. It is funded by Aalto University, departments and schools, and the Academy of Finland. Perhaps a better description would be Aalto HPC (high-performance computing).

Science-IT is the “legal representation” of Aalto Scientific Computing within Aalto.

Computational research is one of the focus areas in Aalto University. The Science-IT project was founded in 2009 to facilitate the computational infrastructure needed in top-tier scientific research. Many Centres of Excellence and departments at Aalto University are using our resources with great success. There are many. Science-IT is administered from the School of Science, and direct Aalto level funding enables use of our resources from all Aalto University, free of charge.

Our services

In Science-IT, we concentrate on mid-range computing and special resources needed by researchers in the School of Science. With local resources, we can provide high-quality support and even research-project-level customization. Because our resources are integrated into the Aalto IT environment, with regular local training in the scientific computing practice to entry-level users, our resources enjoy an ease of access and lower barrier to entry than, for example, CSC HPC resources. We are also a basic research infrastructure, enabling the integration of separately purchased resources to our cluster and storage environments, with dedicated access for the purchaser.

Membership

Departments and schools can join the Science-IT project and receive a share of our resources and dedicated staff support. Please contact Mikko Hakala for details.

Science-IT Management


Operational team: Mikko Hakala, D.Sc. (Tech), Ivan Degtyarenko, D.Sc. (Tech), Richard Darst (Ph.D.), Simo Tuomisto (M.Sc), Enrico Glerean (Ph.D).

To get additional information or how to get involved please contact one of the board member above (first-name.lastname@aalto.fi).

Science-IT is the organizational manifestation of Aalto Scientific Computing.
Science-IT concentrates on mid-range computing and special resources needed by researchers in the School of Science. With local resources, we can provide high-quality support and even research-project-level customization. Because our resources are integrated into the Aalto IT environment, with regular local training in the scientific computing practice to entry-level users, our resources enjoy an ease of access and lower barrier to entry than, for example, CSC HPC resources. We are also a basic research infrastructure, enabling the integration of separately purchased resources to our cluster and storage environments, with dedicated access for the purchaser.

Our team is mainly known for providing the Triton cluster, a mid-range HPC cluster with ~10000 CPUs, 5PB storage capacity, Infiniband network, and ~150 NVIDIA GPUs for deep learning and artificial intelligence research. We provide a Jupyter Notebook based interface to enable light computing with less initial knowledge required to make our services easily accessible to everyone. Our team also works with the CS, NBE, and PHYS departments to provide HT-Condor (high throughput computing), data storage, and a seamless computational research experience. We maintain http://scicomp.aalto.fi, the central hub for scientific computing instructions and have a continuous training program, Scientific Computing in Practice.

Computer Science, Physics, and Neuroscience and Biomedical Engineering

These departments are members of Science-IT, and their local IT staff provide a great deal of scientific computing support, and in fact all the Science-IT team above is contained here. These departments resources are seamlessly integrated with Aalto’s HPC resources.

Computer Science IT

Computer Science IT provides advanced computing, data, and IT services to the Department of Computer Science. Ten years ago, we focused on daily infrastructure and devices. We still do that, but our we now serve a far broader mission including teaching and services, data management, specialised research tools, and cloud services.

Our services

We:

- Handle daily device and infrastructure needs.
- Develop and maintain department services, such as jupyter.cs.aalto.fi or the department services database lapa.aalto.fi.
- Help co-maintain other platforms developed by researchers or teachers.
- Provide services for managing the department’s research data.
- Provide virtual machines.
- Provide advanced consultation for IT needs for research.

But most basic IT tools are handled by Aalto IT Services, not us. We build on their work and make sure research and teaching goes as quickly as possible.

(also note, we don’t primarily serve CS undergraduate students)
Work for CS-IT

We are always looking for students interested in IT, programming, and system administration. We also are a good place for civil service. The most important prerequisites are a good understanding of Linux and a never-ending desire to learn more. Buzzwords you are likely to become familiar with/useful skills to have:

- Kubernetes, docker, and virtual machines
- Web service development
- Puppet (and Ansible)
- Data and storage systems
- Computer hardware, building high-performance workstations

Contact

You can always drop by room A243 if we are there (not during covid-19, please) or join the daily online garage, or contact us by the email address findable on our internal wiki.

See our members on the About Aalto Scientific Computing page.

Partners

We are a leading member of the Finnish Grid and Cloud Infrastructure (FGCI), a university consortium to support mid-range computing in universities. FGCI, via Academy of Finland research infrastructure grants, funds a large portion of our work. Thus, we maintain ties to most other universities in Finland as well as CSC, the national academic computing center. Through the FGCI, we provide grid computing access across all of Finland and Europe.

Our team overlaps with the Departments of Computer Science, Neuroscience and Biomedical Engineering, and Applied Physics. The IT groups in these departments provide advanced Triton support.

We maintain close collaboration with Aalto University IT Services (ITS). We are not a part of ITS, but work closely with them as the computational arm of IT Services. ITS provides the base which we repackage and build on for many of our services.

Our team maintains ties to Aalto Research and Innovation Services to guide data and research policy. Triton is an Aalto-level research infrastructure. Our staff is involved in research policy making, including ethical, data security, and data management. Our team contains several Aalto Data Agents.

We partner with CodeRefinery, a Nordic consortium to assist in training of scientists, to provide training and support computational competence.

10.2.5 Who we are

This table lists people supporting Scientific Computing at Aalto University who considers themselves a part of ASC. If you want to be added here, let us know. We welcome all contributors. There is no Aalto Scientific Computing, just people who want to make computing better.
## 10.2.6 Scientific outputs

Most of the computationally-intensive research outputs from our member departments use our resources. In addition, at least the CS and NBE departments use our data storage for most big data projects. You may view our research results using research.aalto.fi (Science-IT infrastructure section).

## 10.2.7 Current research areas

Our users come from countless research areas:

- Method development
- Computational materials research
- Network research
- Neuroscience
- Data mining
- Deep learning and artificial intelligence
- Big data analysis

## 10.2.8 Other

### Web accessibility

This website is partially conformant with the Web Content Accessibility Guidelines (WCAG) level AA.

This is the accessibility statement for the scicomp.aalto.fi website. The accessibility requirements are based on the Act on the Provision of Digital Services (306/2019).
But as we know from other Aalto web sites, web accessibility doesn’t mean it’s actually useful for any particular purpose. We strive to make this site actually usable by everyone, and we welcome any contributions to help us with that.

Accessibility status of the website

The Web Content Accessibility Guidelines (WCAG) defines requirements for designers and developers to improve accessibility for people with disabilities. Based on self-assessment with Web Accessibility Evaluation Tool, this website is partially conformant with WCAG 2.1 level AA on computers, tablets, and smartphones. Partially conformant means that some parts of the content do not fully conform to the accessibility standard.

Inaccessible content

Below is a description of known limitations, and potential solutions. Please contact us if you observe an issue not listed below.

Known limitations for scicomp.aalto.fi website:

- Inclusion of PDF documents that might have accessibility issues.

Please follow this issue to track updates and improvements to the accessibility of scicomp.aalto.fi.

Technical specifications

Accessibility of scicomp.aalto.fi website relies on the following technologies to work with the particular combination of web browser and any assistive technologies or plugins installed on your computer:

- HTML
- CSS
- WAI-ARIA

These technologies are relied upon for conformance with the accessibility standards used.

Next steps for improving the accessibility

Please follow this issue to track updates and improvements to the accessibility of scicomp.aalto.fi.

Accessibility feedback

We welcome your feedback on the accessibility of scicomp.aalto.fi website. Please let us know if you encounter accessibility barriers on scicomp.aalto.fi website:

- Phone: +358503841575
- E-mail: scicomp@aalto.fi
Supervisory authority

If you encounter any problems with accessibility on the website, first send your feedback to us. We will respond to your feedback within 14 days.

If you are not satisfied with the response you have received from us, or if our response does not arrive within 14 days, you may file a complaint with the Regional State Administrative Agency for Southern Finland. https://www.saavatettavuusvaatimuksset.fi/oikeutesi/ilmoina-ongelma-saavutettavuudessa/

Contact details of the supervisory authority

Regional State Administrative Agency for Southern Finland
Accessibility monitoring unit
website: https://www.saavatettavuusvaatimuksset.fi
email: saavutettavuus@avi.fi
Phone: +358 (0)9 47001

Release and update information

This accessibility statement was last updated on 26 October 2020.
This website was launched on 15 June 2017.
This accessibility statement is based on a similar statement from Fairdata.fi.

About this site

These docs originally came from the Triton User Guide, but now serves as a general Aalto scientific computing guide. The intention is a good central resources for researchers, kept up to date by the whole community. We can’t do this only by ourselves - we invite everyone to help us. Even pointing out unclear parts sets us on the right path. You can and should join us.

Contributing

This documentation is Open Source (CC-BY 4.0), and we welcome contributions from the Aalto community. The project is run on Github in the repository AaltoSciComp/scicomp-docs.
To contribute, you can always use the normal Github contribution mechanisms: make a pull request, issues, or comments. If you are at Aalto, you can also get direct write access. Make a github issue, then contact us in person/by email for us to confirm.

The worst contribution is one that isn’t made. Don’t worry about making things perfect: since this is in version control, we track all changes and will just fix anything that’s not perfect. This is also true for formatting errors - if you can’t do ReStructuredText perfectly, just do your best (and pretend it’s markdown because all the basics are similar).

When you submit a change, there is continuous testing that will notify you of errors, so that you can make changes with confidence.

Contributing gives agreement to use content under the licenses (CC-BY 4.0 or CC0 for examples).
Requirements and building

To build the docs, run `make html`. You can run `make clean` to build it and report only the errors that would cause a failure.

There is a `requirements.txt` file, but the only real Python dependencies to do basic tests is `sphinx` and `sphinx_rtd_theme` (debian packages: `python-sphinx` and `python-sphinx-rtd-theme`).

HTML output is in `_build/html/index.html`, and other output formats are available as well.

Editing

Look at examples and copy. To add sections, add a new page in a subfolder. Link it from the main Table of Contents (`toctree`) in `index.rst` to have the document appear and be cross-referenced.

You can see a complete example from UiT: source and compiled HTML.

ReStructured text

ReStructured Text is similar to markdown for basics, but has a more strictly defined syntax and more higher level structure. This allows more semantic markup, more power to compile into different formats (since there isn’t embedded HTML), and advanced things like indexing, permanent references, etc.

Restructured text home and quick reference.

Note: Literal inline text uses `` instead of a single ` (second works but gives warning).

A very quick guide is below.

<table>
<thead>
<tr>
<th>Inline code/monospace, emphasis, strong emphasis</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Inline code/monospace</code>, <em>emphasis</em>, <strong>strong emphasis</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block quote</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block quote</td>
</tr>
</tbody>
</table>

::

    Block quote
    Block quote

Block quotes can also start with paragraph ending in double colon, like this:

Block quote

Block quotes can also start with paragraph ending in double colon, like this:

    Block quote
Inline link, or anonymous, or separate, or different text links. Trailing underscores indicate links.

```
Inline `link <https://www.python.org>`_, or
anonymous__, or
separate_, or
'different text <separate_>'_ links.
Trailing underscores indicate links.
___ https://www.python.org
.. _separate: https://www.python.org
```

Linking to the web. If possible use a permanent reference (next section), but you can also refer to specific files by name. Note, that for internal links there are no trailing underscores:

```
:doc:`../tut/interactive.rst` (recommended)
`../tut/interactive.rst` (short, no warning if link breaks)
```

With different text:
```
:doc:`Text <../tut/interactive.rst>` (recommended)
'Text <../tut/interactive.rst>' (short, no warning if link breaks)
```

Internal links. Permanent references across files

Label things this way (note only one colon):

```
.. _label-name:
```

Reference them this way:

```
:ref:`label-name` (recommended)
'label-name' (short, no warning if link breaks)
'Text <label-name>' (short, no warning if link breaks)
```

Notes, warnings, etc.

**Note:** This is a note

**Warning:** This is a warning

```
.. note::
   This **is** a note

.. warning::
   This **is** a warning
```
These docs are open source: all content is licensed under CC-BY 4.0 and all examples under CC0 (public domain). Additionally, this is an open project and we strongly encourage anyone to contribute. For information, see the About this site and the Github links at the top of every page. Either make Github issues, pull requests, or ask for direct commit access. Be bold: the biggest problem is missing information, and mistakes can always be fixed.

- genindex
- search