Hands-on Scientific Computing

Apr 04, 2022

Contents

The transition between courses and exercise and computational research can be difficult - there are so many important things to know that aren't academic, thus they aren't taught in courses. This guide is your starting point - we guide you through the practical tools and tricks that you would otherwise have to figure out on your own or learn from friends.

Hands-on SciComp is a "map" of diverse skills that you need for scientific computing, which are often not directly taught in classes these days. It is the practice

CHAPTER 1

Using the material

This is primarily a self-study course and reference material, which you can browse at your own pace as it becomes relevant to you. A coordinated set of levels (~1 day) and modules (~ 1 hour) splits skills into levels depending on your needs. A course instructor or research supervisor might point you at what is most important for your current work. Then, focus on those levels at your own pace.

This course is coordinated by Aalto University Science-IT (See *About* for contact info)

CHAPTER 2

Study credits

If you are at Aalto University, you can get *study credits*. If you are in Finland but not Aalto, *you can get credits via the free FiTech program*.

CHAPTER 3

Course video introduction

See course video introduction series here

CHAPTER 4

Outline

Level	For who?	Covers what?
A: Basics: What computing and how?	Mini-level for everyone who's doing science with your computer or may need to rely on computing resources later.	What types of resources are available, when you'd use them, and how to get help. How to set up your computer to do scientific work. What comes next.
B: Related science skills	Everyone publishing in a somewhat computational field.	Making figures, papers, posters, and so on they way it's done in computational fields.
C: Scientific computing (Linux and shell)	Everyone who's doing more than pointing and clicking single applications on your own computer or needs more computing power.	In this level, you learn how to extend your power beyond your own computer or existing applications. Includes data management, scripting, Linux, and servers. Linux and the shell are a major point here: this is the defacto (and only) good way to increase power. Equal to the B level.
D: Clusters and high-performance computing	Those who need more power than their own computer and need to move to a cluster, whether or not it's highly parallelized.	Computing on clusters and remote servers, more advanced Linux, more scripting, batch systems, HPC data management.
E: Scientific coding	When you start writing your own software to do your research.	Version control, how to manage code, software, and data even more. We don't cover programming itself, just the untaught parts about how to use it as a researcher. Equal to the D level.
F: Advanced high performance computing	Those who are programming the most demanding parallel scientific applications.	MPI (message passing interface, a parallel programming framework), OpenMP (another one), GPU programming, etc. And anything more advanced.

We have material for different learning styles: you might prefer to watch a video to see quick live examples, or read something for more detail. All of these aspects compliment each other, and you can do what suits you the best.

4.1 A: Basics

What's available? How can it be found? What basic things do you need to install?

4.1.1 A: Basics

A01 Introduction to scientific computing

Description	Get started with common scientific computing guidelines.
Video intro	
Reading	>Good computing practices for everyone regardless of skills
Questions	>What kind of a workflow to follow? >Where to get help?
Aalto	> Welcome, Researchers > Getting help

A10 Configuring Linux for scientific work

Description	Linux is great for scientific work. This goes over some key things to install to get going.
Video intro	
Reading	>Software Carpentry set up material
Questions	>What kind of tools do I need?
Aalto	

About Linux

Linux is an operating system known for its flexibility and power. It doesn't hide things from the user, which makes it especially suitable for scientific computing, where you need to assemble your own pieces together and have full control. Because of it's open-source spirit, many other open-source tools are developed for it

Linux is not just one thing: there are many **distributions** which combine software. Which one to choose is basically user preference (ask your friends what they use), but there are two major types: **Debian-based** (uses `apt` to install programs) and **Red-hat based** (uses `yum` to install programs). In practice, Ubuntu is a good default these days. These instructions (so far) are for Debian-based distributions like Ubuntu.

On Ubuntu, the standard way to install things is `sudo apt-get install $package_names ...`

Shell tools

The shell provides an interface to efficiently access the true power of a computer. Now we use it to install tools but it can be used for many other tasks too.

Every Linux distribution comes with a shell already installed. Start the "Terminal" or "Shell" to see it. To verify, try running this:

```
$ echo $SHELL
/bin/bash
```

The convention is the `$` represents lines you type (without the `$` - notice most shell prompts have it there already), the other lines are what comes out. `#` represents comments.

If you want a crash course on using the shell, see the Aalto shell crash course. You don't need this right now.

Version control (git)

Using version control is like an insurance for your projects. It is not only about tracking changes but also to improve your project visibility and make it easier to collaborate.

Git is the most popular system for version control and GitHub is one of the services that provide online storing for projects.

This comes included in all operating systems, but needs to be installed. Here, we install git and some other useful frontends for it:

```
$ sudo apt install git gitk gitg
```

Verify from the shell (see above to start the shell):

```
$ git --version
git version 2.20.1
```

Your organization might provide you access to some other repository manager than GitHub but since GitHub is a higher availablity solution, it does not hurt to create an account there. You can sign up for Github here

Anaconda (Python)

In software development there are some standard packages that are useful to have without the trouble of installing them separately with their dependencies.

There are very many programming languages, and you probably won't only use Python. But, it is quite common so we mention it here. We install the Anaconda distribution of Python: it gets you all the basic things you need, and can also install R and other programming languages, too. Anaconda is large and has all the most common tools people need - if you want to save space, install Miniconda instead (then you have to decide what extra packages you want).

- Anaconda
- Miniconda

This will get you Jupyter and many other Python things, too.

Anaconda allows you to manage your development environment which is good since you can have different environments dedicated to their designated purposes.

Todo: How to install it in the shell. How to start/use it. Easier install instructions. Link the SWC video.

To verify from the shell (see above to start the shell):

```
$ python3 -V
Python 3.6.8 :: Anaconda custom (64-bit)

$ conda info
     active environment : None
...
       base environment : /home/rkdarst/anaconda3  (writable)
```

Editor

It's good to have one command-line editor and one graphical Integrated Development Environment.

Command line editor

For fast things, you want to be able to edit files quickly from a the command line. `Nano` is the simplest to use. If you want, you can check out vim or emacs, but they certainly harder to use so we don't recommend them to start off.

To install `nano`:

```
$ sudo apt-get install nano
```

Todo: Is this the most useful verification?

See this nano tutorial to learn more. To verify nano from the shell (see above to start the shell):

```
$ nano my_file.txt
```

Integrated Development Environment

** You should install one good **Integrated Development Environment (IDE)**. This has coding, version control, and many more things build in to one interface. These days, **VSCode** is the most popular. Install from the vscode website. Out of principle, we recommend you disable data collection.

Emacs can also serve as an IDE once you learn enough about it.

Jupyter

Jupyter is an interactive way to explore data and do programming. It can be used to add code, output, titles, text and visualisations into one document. It's already installed along with Anaconda. To start it in a certain directory, go to that directory in the shell and run:

```
$ jupyter notebook       # older notebook interface
$ jupyter lab            # newer JupyterLab interface
```

Follow this to install useful extensions to your environment. Especially ipywidgets are needed if you continue to do exercises.

Other programming tools

Install:

```
$ sudo apt install build-essential meld
```

- `build-essential` installs some basic compilers and so on.
- `meld`: A graphical diff program

If you wish to obtain credits from the course, you might need

- NumPy
- Matplotlib

to complete exercises. These libraries are pre-installed with Anaconda installation. Further information about installations can be found here: NumPy and Matplotlib

A11 Configuring Mac for scientific work

Description	Get your Mac computer set up for scientific computing tasks.
Video intro	>Software Carpentry tutorial for Shell, Git and Nano installations on a Mac.
Reading	>Software Carpentry set up material
Questions	
Aalto	

MacOS became popular for scientific work when it became based on Unix: it provided an easy interface *and* the shell, which is a great combination.

This page gets you set up for basic scientific work using Python.

Shell tools

The shell provides an interface to efficiently access the true power of a computer. Now we use it to install tools but it can be used for many other tasks too.

Mac comes with the bash (or zsh for 10.15 and later), so you don't need to do anything. Just start it by TODO. To verify, try running this:

```
echo $SHELL
```

Version control (git)

Using version control is like an insurance for your projects. It is not only about tracking changes but also to improve your project visibility and make it easier to collaborate.

Git is the most popular system for version control and GitHub is one of the services that provide online storing for projects.

You install Git for MacOS by downloading the most recent "mavricks" installer from http://sourceforge.net/projects/git-osx-installer/files/

If you have Homebrew (a package manager) you can do:

```
brew install git
```

Nothing appears in Applications, since it's a command line program. From

Verify it from the shell terminal:

```
git --version
```

Your organization might provide you access to some other repository manager than GitHub but since GitHub is a higher availablity solution, it does not hurt to create an account there. You can sign up for Github here

Anaconda (Python)

In software development there are some standard packages that are useful to have without the trouble of installing them separately with their dependencies.

There are very many programming languages, and you probably won't only use Python. But, it is quite common so we mention it here. We install the Anaconda distribution of Python: it gets you all the basic things you need, and can

also install R and other programming languages, too. Anaconda is large and has all the most common things people need - if you want to save space, install Miniconda instead (then you have to decide what extra packages you want).

- Anaconda
- Miniconda

This will get you Jupyter and many other Python things, too.

Anaconda allows you to manage your development environment which is good since you can have different environments dedicated to their designated purposes.

Todo: Same stuff from Linux page. How to use it.

To verify from the shell (see above to start the shell):

```
$ python3 -V
Python 3.6.8 :: Anaconda custom (64-bit)

$ conda info
     active environment : None
...
       base environment : /home/rkdarst/anaconda3  (writable)
```

Homebrew

Homebrew is a package manager for MacOS, which lets you install lots of packages easily. Many of these are essential to having a good environment for programming, and taking full advantage of MacOS.

To install, go to brew.sh and follow instructions. You can then, for example, use `brew install` to install many things you may need.

After installing, you can run `brew doctor` to ensure everything was installed correctly.

Editor

FOR IDE (Integrated development environment): Visual Studio Code is a free editor available for Windows, macOS and Linux. It is a good alternative for both a beginner and a more advanced user as it is simple to use but highly customizable. Install and learn more here. Out of principle, we recommend you disable data collection.

For command line: You should make sure `nano` is installed by typing in the shell for instance, `nano my_file.txt`. You can also use vi/vim or emacs but as those are harder to use, we do not recommend them for your first command line editor. Nano is used through keyboard shortcuts and some of them are shown in the editor. See this tutorial to start editing with nano.

Jupyter

Jupyter is an interactive way to explore data. It can be used to add code, output, titles, text and visualisations into one document. It's already installed along with Anaconda.

Follow this to install useful extensions to your environment. Especially ipywidgets are needed if you continue to do exercises.

Other programming tools

If you wish to obtain credits from the course, you might need

- NumPy
- Matplotlib

to complete exercises. These libraries are pre-installed with Anaconda installation. Further information about installations can be found here: NumPy and Matplotlib

A12 Configuring Windows for scientific work

Description	Get your Windows computer set up for scientific computing tasks.
Video intro	>Software Carpentry Git Bash tutorial for Windows.
Reading	>Software Carpentry set up material
Questions	
Aalto	

About Windows

Windows is perhaps the most common operating system for desktop computers, but historically hasn't been that common or good for scientific work. However, this is changing and these days you can do a lot of good stuff with Windows if you set it up right. We'll walk through it here.

Shell tools

The shell provides an interface to efficiently access the true power of a computer. Now we use it to install tools but it can be used for many other tasks too.

Windows comes with CMD (`cmd.exe`) known as command prompt. You can find CMD by typing `cmd` in your start menu search bar. A slightly better alternative would be a Git bash command line because Windows command prompt does not support many UNIX commands. Git Bash emulates a bash environment and lets you use all git features plus most of standard unix commands - so you are immediately compatible with Mac and Linux.

See the next section for installation instructions.

Version control (git)

Using version control is like an insurance for your projects. It is not only about tracking changes but also to improve your project visibility and make it easier to collaborate.

To install Git Bash, follow this tutorial made by Software Carpentry. You only need to follow the video instructions for Git Bash (until 2:50) because the newest versions of Git Bash should install the needed *nix environment tools automatically.

Please note that the Git setup window will ask you to choose your default text editor and it will first suggest vi/vim. However, we do not recommend vi/vim for your first command line editor but rather to change it to nano text editor, which is more easier for a beginner to use.

After you are all set up, open your Git Bash and try it out by typing for example: `nano` and `git --version`

Links:

- Git Bash
- Nano text editor tutorial
- Git can also be installed through Anaconda

Your organization might provide you access to some other repository manager than GitHub but since GitHub is a higher availablity solution, it does not hurt to create an account there. You can sign up for Github here

Anaconda (Python)

In software development there are some standard packages that are useful to have without the trouble of installing them separately with their dependencies.

There are very many programming languages, and you probably won't only use Python. But, it is quite common so we mention it here. We install the Anaconda distribution of Python: it gets you all the basic things you need, and can also install R and other programming languages, too. Anaconda is large and has all the most common tools people need - if you want to save space, install Miniconda instead (then you have to decide what extra packages you want).

- Anaconda
- Miniconda

Copy other information from https://coderefinery.github.io/installation/python/

Anaconda allows you to manage your development environment which is good since you can have different environments dedicated to their designated purposes.

Todo: We need to give details about how to use it.

Editor

For IDE (Integrated development environment): Visual Studio Code by Microsoft is a free source code editor. It offers customizable functionalities for a more advanced user but is simple enough for a beginner to start with. Install and learn more here.

Other good alternative for Windows is Notepad++ source code and text editor. Notepad++ is not exactly an IDE as it lacks features that IDEs have but plugins are available to add functionalities. Download and read more here.

Jupyter

Jupyter is an interactive way to explore data. It can be used to add code, output, titles, text and visualisations into one document. It's already installed along with Anaconda. To start it in a certain directory, go to that directory in the shell and run:

```
$ jupyter notebook       # older notebook interface
$ jupyter lab            # newer JupyterLab interface
```

Follow this to install useful extensions to your environment. Especially ipywidgets are needed if you continue to do exercises.

Other programming tools

For remote network tools: MobaXterm

If you wish to obtain credits from the course, you might need

- NumPy
- Matplotlib

to complete exercises. These libraries are pre-installed with Anaconda installation. Further information about installations can be found here: NumPy and Matplotlib

	About	Questions	Video Intro	Reading	Aalto
A01 Introduction to scientific computing	Get started with common scientific computing guidelines.	>What kind of a workflow to follow? >Where to get help?		>Good computing practices for everyone regardless of skills	> Welcome, Researchers > Getting help
A10 Configuring Linux for scientific work	Linux is great for scientific work. This goes over some key things to install to get going.	>What kind of tools do I need?		>Software Carpentry set up material	
A11 Configuring Mac for scientific work	Get your Mac computer set up for scientific computing tasks.		>Software Carpentry tutorial for Shell, Git and Nano installations on a Mac.	>Software Carpentry set up material	
A12 Configuring Windows for scientific work	Get your Windows computer set up for scientific computing tasks.		>Software Carpentry Git Bash tutorial for Windows.	>Software Carpentry set up material	

	About	Questions	Video Intro	Reading	Aalto
A01 Introduction to scientific computing	Get started with common scientific computing guidelines.	>What kind of a workflow to follow? >Where to get help?		>Good computing practices for everyone regardless of skills	> Welcome, Researchers > Getting help
A10 Configuring Linux for scientific work	Linux is great for scientific work. This goes over some key things to install to get going.	>What kind of tools do I need?		>Software Carpentry set up material	
A11 Configuring Mac for scientific work	Get your Mac computer set up for scientific computing tasks.		>Software Carpentry tutorial for Shell, Git and Nano installations on a Mac.	>Software Carpentry set up material	
A12 Configuring Windows for scientific work	Get your Windows computer set up for scientific computing tasks.		>Software Carpentry Git Bash tutorial for Windows.	>Software Carpentry set up material	

4.2 B: Related science skills

Assorted things that help you with your work, but not directly related to doing computations.

4.2.1 B: Related science skills

B21 Jupyter Notebooks

Description	Notebooks are an efficient way to make self-documenting code and scripts and make data science a bit easier.
Video intro	>Data analysis with Jupyter video series.
Reading	>CodeRefinery Jupyter course
Questions	>For what kind of work Jupyter suits the best? >How is a Jupyter kernel launched?
Aalto	JupyterHub for students/misc work, JupyterHub for HPC

Jupyter Notebook is part of the Project Jupyter together with JupyterHub and JupyterLab. It provides an online environment for creating documents which can contain executable code, explanatory text and other resources (e.g. graphs).

Jupyter Notebook is good for teaching, demonstrating and sharing ideas and testing out examples in browser. With a large codebase and more advanced projects it is better to seek out other options as version control and automated testing become harder.

Installation

If you have installed Anaconda, Jupyter should already be installed on your machine. See that your Jupyter installation is working by typing `jupyter-notebook` in the shell. JupyterLab can be launched by typing `jupyter-lab`.

See here for optional extensions you can install in the notebooks.

	About	Questions	Video Intro	Reading	Aalto
B20 Data Management	Data needs to be organized and handled well, or else it quickly becomes unusable. There are good and bad ways to do this.	>How to use version control in data management? >What kind of ordering makes the best structure?	>What is not data management, in 5 minutes and >Data structuring	>The Turing Way data management chapter >Information for research at FSD	Research data management at Aalto, at scicomp.aalto.fi
B21 Jupyter Notebooks	Notebooks are an efficient way to make self-documenting code and scripts and make data science a bit easier.	>For what kind of work Jupyter suits the best? >How is a Jupyter kernel launched?	>Data analysis with Jupyter video series.	>CodeRefinery Jupyter course	JupyterHub for students/misc work, JupyterHub for HPC
B30 Making figures	How to make publication-quality figures for your work.	>What kinds of tools exist for making figures and plots? >What are some dos and don'ts for plots and figures?	>Inkscape tutorials covering Inkscape basics for making figures, flowcharts, etc. >A series of Matplotlib tutorials for plotting in Python. >ggplot2 tutorials in R.	>PLOS guidelines for better figures	
B31 LaTeX for scientific publications	LaTeX is the standard method for making publications in the computational and physical sciences.	>How to build a LaTeX document?	>LaTeX tutorial series covering LaTeX formatting and constructing a report	>Introduction to LaTeX (online book) >Short summary of LaTex features	> Guide to LaTeX (FI)
B32 Scientific posters	Making a scientific poster is a common task, but not often taught. There are better tools than PowerPoint.	>What makes a clear and concise poster?	>The dos and don'ts of making a scientific poster	>Basics of creating a Research Poster	

	About	Questions	Video Intro	Reading	Aalto
B20 Data Management	Data needs to be organized and handled well, or else it quickly becomes unusable. There are good and bad ways to do this.	>How to use version control in data management? >What kind of ordering makes the best structure?	>What is not data management, in 5 minutes and >Data structuring	>The Turing Way data management chapter >Information for research at FSD	Research data management at Aalto, at scicomp.aalto.fi
B21 Jupyter Notebooks	Notebooks are an efficient way to make self-documenting code and scripts and make data science a bit easier.	>For what kind of work Jupyter suits the best? >How is a Jupyter kernel launched?	>Data analysis with Jupyter video series.	>CodeRefinery Jupyter course	JupyterHub for students/misc work, JupyterHub for HPC
B30 Making figures	How to make publication-quality figures for your work.	>What kinds of tools exist for making figures and plots? >What are some dos and don'ts for plots and figures?	>Inkscape tutorials covering Inkscape basics for making figures, flowcharts, etc. >A series of Matplotlib tutorials for plotting in Python. >ggplot2 tutorials in R.	>PLOS guidelines for better figures	
B31 LaTeX for scientific publications	LaTeX is the standard method for making publications in the computational and physical sciences.	>How to build a LaTeX document?	>LaTeX tutorial series covering LaTeX formatting and constructing a report	>Introduction to LaTeX (online book) >Short summary of LaTex features	> Guide to LaTeX (FI)
B32 Scientific posters	Making a scientific poster is a common task, but not often taught. There are better tools than PowerPoint.	>What makes a clear and concise poster?	>The dos and don'ts of making a scientific poster	>Basics of creating a Research Poster	

4.3 C: Linux and shell

The basics which everything else is built on.

4.3.1 C: Linux and shell

	About	Questions	Video Intro	Reading	Aalto
C10 Basic shell	Let's face it: the command line is the basis of most data science and programming.	>How does the shell work? >When to use a CLI instead of a GUI?	>Shell crash course	>Shell crash course >Software carpentry Shell-novice >The first part of our shell course is good too.	
C23 Text editors and IDEs	Your best friend is a good text editor - sometimes you just need to edit things quickly on some remote system.	>Which tools to use for code development and editing?	>Get to know VS Code tutorial series	>Software Carpentry shell-novice, "Create a text file" part of section 3 >Tutorial on IDEs by CodeRefinery.	
C20 Shell scripting	If you can do it on the Linux shell, you can automate it.	>How to make use of shell scripting tools in repetitive task automation?	>Shell scripting tutorials.	>Continue with the Science-IT Linux shell tutorial part 2.	
C21 Version control for you	Version control lets you track changes, go back in time, and collaborate on code and papers: an absolute requirement for scientific computing.	>What is Git? >How to initialize a Git repository?	>Why use version control >Git for beginners	>Introduction to version control by CodeRefinery	
C22 SSH and remote access	A short but important course: how to do work remotely. Different expert tips for making ssh better, too.	>What does SSH mean and when to use it?	>Introduction to secure shell by Software Carpentry	>SSH for working on a remote machine.	How to make ssh work better by Aalto Scicomp
C23 Make	Automate the repetitive stuff with Make.	>How can a Makefile be useful in your large project?	>Episodes on Make by Software Carpentry	>Short introduction on what is a Makefile and basic operations. >For more information on Makefiles see GNU Make Manual	

	About	Questions	Video Intro	Reading	Aalto
C10 Basic shell	Let's face it: the command line is the basis of most data science and programming.	>How does the shell work? >When to use a CLI instead of a GUI?	>Shell crash course	>Shell crash course >Software carpentry Shell-novice >The first part of our shell course is good too.	
C23 Text editors and IDEs	Your best friend is a good text editor - sometimes you just need to edit things quickly on some remote system.	>Which tools to use for code development and editing?	>Get to know VS Code tutorial series	>Software Carpentry shell-novice, "Create a text file" part of section 3 >Tutorial on IDEs by CodeRefinery.	
C20 Shell scripting	If you can do it on the Linux shell, you can automate it.	>How to make use of shell scripting tools in repetitive task automation?	>Shell scripting tutorials.	>Continue with the Science-IT Linux shell tutorial part 2.	
C21 Version control for you	Version control lets you track changes, go back in time, and collaborate on code and papers: an absolute requirement for scientific computing.	>What is Git? >How to initialize a Git repository?	>Why use version control >Git for beginners	>Introduction to version control by CodeRefinery	
C22 SSH and remote access	A short but important course: how to do work remotely. Different expert tips for making ssh better, too.	>What does SSH mean and when to use it?	>Introduction to secure shell by Software Carpentry	>SSH for working on a remote machine.	How to make ssh work better by Aalto Scicomp
C23 Make	Automate the repetitive stuff with Make.	>How can a Makefile be useful in your large project?	>Episodes on Make by Software Carpentry	>Short introduction on what is a Makefile and basic operations. >For more information on Makefiles see GNU Make Manual	

4.4 D: Clusters and High Performance Computing

Using advanced computational resources. This will be highly site-specific. We include some basic information here, but you will always have to refer to specific site's instructions.

4.4.1 D: Clusters and High Performance Computing

	About	Questions	Video Intro	Reading	Aalto
D01 What is HPC?	Before you can use larger resources, you need to understand the difference from your own computers	>What are the scales of computing?		>HPC Intro	Triton cluster intro
D20 Modules and software	Using and installing software on a cluster is different from your own computer, because hundreds of people are sharing it. Modules are the solution.	>How do you use `module`? >How do you find software?	>Lmod introduction	>Triton tutorials for intro: modules, applications, >Lmod user guide	> Software and applications, > modules
D21 Batch systems	On a cluster, you have to share resources with others. Slurm is one batch queuing system that makes it possible.	>What role does the batch system fill? >How does one submit to the batch system?	>Slurm basics >interactive jobs >batch jobs	Triton tutorials: >interactive, >serial, >array	Triton tutorials: interactive, serial, array
D22 HPC Storage	Storage turns out to be just as important as computing power. There are different places available, each with different advantages.	>Why is storage so important? >How can you monitor input/output (I/O) performance? >How to best handle your data?	>HPC I/O principles	>Storage basics.	Triton tutorials: storage basics. More advanced: lustre, local storage, small files
D23 Parallel computing	The point of a cluster is to run things in parallel. Shared memory (OpenMP) and message passing (MPI) are the most common models. Learn how to run them, not write them.	>What are the main models of parallel code? >How are they run on clusters? >How do you figure out what your code uses?		>Parallel jobs.	Triton tutorials: parallel.
D24 Advanced shell scripting and automation	Hands-on shell scripting, putting everything together to automate large computations on the cluster.			Various courses, finishing the linux shell tutorial is a good start. The Advanced bash scripting guide is a classic.	

	About	Questions	Video Intro	Reading	Aalto
D01 What is HPC?	Before you can use larger resources, you need to understand the difference from your own computers	>What are the scales of computing?		>HPC Intro	Triton cluster intro
D20 Modules and software	Using and installing software on a cluster is different from your own computer, because hundreds of people are sharing it. Modules are the solution.	>How do you use `module`? >How do you find software?	>Lmod introduction	>Triton tutorials for intro: modules, applications, >Lmod user guide	> Software and applications, > modules
D21 Batch systems	On a cluster, you have to share resources with others. Slurm is one batch queuing system that makes it possible.	>What role does the batch system fill? >How does one submit to the batch system?	>Slurm basics >interactive jobs >batch jobs	Triton tutorials: >interactive, >serial, >array	Triton tutorials: interactive, serial, array
D22 HPC Storage	Storage turns out to be just as important as computing power. There are different places available, each with different advantages.	>Why is storage so important? >How can you monitor input/output (I/O) performance? >How to best handle your data?	>HPC I/O principles	>Storage basics.	Triton tutorials: storage basics. More advanced: lustre, local storage, small files
D23 Parallel computing	The point of a cluster is to run things in parallel. Shared memory (OpenMP) and message passing (MPI) are the most common models. Learn how to run them, not write them.	>What are the main models of parallel code? >How are they run on clusters? >How do you figure out what your code uses?		>Parallel jobs.	Triton tutorials: parallel.
D24 Advanced shell scripting and automation	Hands-on shell scripting, putting everything together to automate large computations on the cluster.			Various courses, finishing the linux shell tutorial is a good start. The Advanced bash scripting guide is a classic.	

4.5 E: Scientific coding

This isn't about doing the programming itself, but managing it in research projects. A prerequisite is knowing some programming language already.

4.5.1 E: Scientific coding

	About	Questions	Video Intro	Reading	Aalto
E60 Modular code development	Break your large programs into small problems by separating aspects of desired functionality to different sub-modules.	>How to divide code into independent modules? >What are pure functions like?	>Python example of breaking code into small components	>Lesson on Modular code development by CodeRefinery	
E61 Software testing	It is important to ensure that your program performs effectively and without failures. Adding tests for your software can save a lot of your time later.	>How to test code on different levels? >What kind of testing tools are there?	>Software testing fundamentals by Software Carpentry	>Lesson on testing by CodeRefinery	
E62 Profiling	Code efficiency is critical especially in HPC. Learn to measure the performance of your programs.	>What is profiling used for?	>Profiling Python code with cProfile	>Profiling tools for Linux >Profiling for C and Python >An intro article on Ruby and Python's profilers	Triton profiling guide
E63 Debugging	Detect, investigate and resolve bugs.	>How to debug different types of errors?	>Debugging strategies	>Debugging in a nutshell. >See Triton's debugging guide >A hands-on tutorial on pdb debugger	
E02 Software Licensing	Sharing your work can be very beneficial. Take a look at social coding and software licensing.	>What is free software? >Why should you share your code?	>Brief introduction to differences between open and closed source software	>Lesson on social coding by CodeRefinery >Brief guide to licensing	
E04 Documentation	Document your project so other people can easily use the code and even contribute to it.	>What should be included in a documentation?	>Documentation with Sphinx	>Tools for documentation >CodeRefinery lesson on documentation	
E03 Reproducible research	How different tools can improve reproducibility.	>Which tools can help with reproducibility?	>What is reproducible research	>Lesson by CodeRefinery	

	About	Questions	Video Intro	Reading	Aalto
E60 Modular code development	Break your large programs into small problems by separating aspects of desired functionality to different sub-modules.	>How to divide code into independent modules? >What are pure functions like?	>Python example of breaking code into small components	>Lesson on Modular code development by CodeRefinery	
E61 Software testing	It is important to ensure that your program performs effectively and without failures. Adding tests for your software can save a lot of your time later.	>How to test code on different levels? >What kind of testing tools are there?	>Software testing fundamentals by Software Carpentry	>Lesson on testing by CodeRefinery	
E62 Profiling	Code efficiency is critical especially in HPC. Learn to measure the performance of your programs.	>What is profiling used for?	>Profiling Python code with cProfile	>Profiling tools for Linux >Profiling for C and Python >An intro article on Ruby and Python's profilers	Triton profiling guide
E63 Debugging	Detect, investigate and resolve bugs.	>How to debug different types of errors?	>Debugging strategies	>Debugging in a nutshell. >See Triton's debugging guide >A hands-on tutorial on pdb debugger	
E02 Software Licensing	Sharing your work can be very beneficial. Take a look at social coding and software licensing.	>What is free software? >Why should you share your code?	>Brief introduction to differences between open and closed source software	>Lesson on social coding by CodeRefinery >Brief guide to licensing	
E04 Documentation	Document your project so other people can easily use the code and even contribute to it.	>What should be included in a documentation?	>Documentation with Sphinx	>Tools for documentation >CodeRefinery lesson on documentation	
E03 Reproducible research	How different tools can improve reproducibility.	>Which tools can help with reproducibility?	>What is reproducible research	>Lesson by CodeRefinery	

4.6 F: Advanced high performance computing

Assorted advanced topics which we can't go into details of, but might be interesting to you.

4.6.1 F: Advanced high performance computing

	About	Questions	Video Intro	Reading	Aalto
Fxx Parallel programming computers	This is an academic course taught in the CS department. It mainly covers OpenMP and CUDA. Usually taught in 5th period (Apr-May), search MyCourses/Oodi for CS-E4580.				
Fxx GPU Programming	This was an advanced guest course, useful if you want to know how to program GPU applications.			>Materials here	
Fxx MPI Programming	This was an advanced guest course, useful if you want to know internals of MPI or program MPI applications.			>Materials here	
Fxx HTCondor	Condor allows you to use many workstations as a high throughput cluster, ideal for mid-range embarrassingly parallel problems.			>Materials here	

	About	Questions	Video Intro	Reading	Aalto
Fxx Parallel programming computers	This is an academic course taught in the CS department. It mainly covers OpenMP and CUDA. Usually taught in 5th period (Apr-May), search MyCourses/Oodi for CS-E4580.				
Fxx GPU Programming	This was an advanced guest course, useful if you want to know how to program GPU applications.			>Materials here	
Fxx MPI Programming	This was an advanced guest course, useful if you want to know internals of MPI or program MPI applications.			>Materials here	
Fxx HTCondor	Condor allows you to use many workstations as a high throughput cluster, ideal for mid-range embarrassingly parallel problems.			>Materials here	

4.6.2 About

Hands-on Scientific Computing is a guide for all researchers and students who have demanding computing needs.

- It can be browsed as needed or as a reference for people working independently.
- It can (in the future) be used as a self-study course to bridge the gap between academic study and independent research.
- You do not have to enroll in this course unless you wish to be graded.

Hands-on Scientific Computing is a course born out of Aalto University Science-IT, CodeRefinery, and many other inspirations. Initial funding is provided by an Aalto Online Learning grant.

Support

If you have questions about Hands-on SciComp in general, see CodeRefinery zulipchat.

You can also contact us via **scip@aalto.fi**

Design and development

This is an open project, which means we encourage contributions from everyone and also accept that we have some compromises to make it generally useful. For local site customizations, we have a templating system using the `site/` directory, so that we can have the same source but every site can have their local customizations.

The CONTRIBUTING file explains in easy terms how to contribute specific things.

The DESIGN file explains how we structure the levels, modules, and information within the modules.

The README file lists technical information about contributing.

Partners and users

- Aalto University Science-IT - lead

4.6.3 Study credits (Aalto)

See also:

If you are in Finland but not Aalto, *you can get credits via the free FITech program*.

You can earn credits from completing this course by doing exercises (if you are in Finland at least). One credit comes from completing exercises for levels A, B and C. A second credit can be earned by completing exercises for D and E.

Exercises (and this course in general) are designed to make you experience these tools, but are only a starting point to exploration.

Instructions if you are at Aalto University:

- The course material is on this page, and can be browsed at your own pace.
- Log in to the exercise system kept separately. You need to attempt at least 90% of the exercises and pass at least 50%. You may do this at your own pace.
- Note that exercises are subject to change, even though the course is continuous the exercises may change if you delay completing the course for too long.
- Request grading and by the instructions in section 1.1.

4.6.4 Study credits (FITech)

See also:

If you are already a student at Aalto University, you should use the simpler procedure at *Study credits (Aalto)*.

How to start

Hands-on scientific computing is a self-study course where you can reflect on your needs and determine what is useful for you. Modules have been structured from A to F in a way that the difficultness level rises gradually.

If you only wish to browse the materials, you can freely do so. If you wish to acquire credits or a certificate, you should read on.

Note: If you are at Aalto University, these steps are not needed: you can log in to the exercise system with your existing Aalto account and request credits from scip@aalto.fi directly.

Exercises

Our exercise page can be found here: A+ Hands-on scientific computing.

Logging in is required to submit the exercises, so you should log in as "External to Aalto".

Course evaluation is divided into modules A-C and D-E and you can earn 1 credit from each modules (overall 2 credits). Instead of credits users may also request a certificate after approved completion of course.

More information about the grading practicalities and the exercises can be found on the exercise page.

Although the course is an always open course, notice that the content and exercises might change over time and your performance may not be valid anymore. Please make sure to finish your work in six months time to secure successful completion of the course.

Applying for credits

Please fill in the FITech application at Studyinfo.fi:

a. Application for degree students from other Finnish Universities (Bachelor's or Master's students)

b. Application for FITech adult learners (PhD students and adult learners)

Make sure you are using the same personal information during the course and in the application. You need to have Finnish personal identity code to apply for the credits and online banking access code for strong authentication to confirm your identity. You will receive automatic messages from the application system. After the application has been handled, you need to confirm your identity. When everything is in order and grade registered, we will send you the electronic transcript of records. Questions? Please contact fitech-sci@aalto.fi.

After completing the exercises, you should notify **fitech-sci@aalto.fi** that your performance is ready for evaluation. If you have successfully met the grading criteria (90% of exercises completed and have at least 50% correct), your performance will be accepted and credits registered.

4.6.5 For teachers

If you are here, you are probably a teacher of some course that requires basic practical computational skills (Linux command line, git, shell scripting, etc), but you don't want to have to teach that yourself. We have the solution for you here, which can be used in different ways:

- Send your students here when it is clear they are missing some prerequisites
- Add this course as a soft prerequisite/recommended reading
- You contribute new material here, instead of making your own prerequisite documents.

Our philosophy is "all the basics someone needs to know is online somewhere", so we help people find that rather than try to make our own material. This also helps people become more self-sufficient in the future.

Sample text

This could be added to your course prerequisites/syllabus, for example:

> To succeed in this course, it is best to have some practical skills in {Linux and command line work, lightweight software development tools, ... }. You can review this at https://hands-on.coderefinery.org/. For this course, the {C and D levels are important for success. You should especially focus on C10 Basic shell, C23 Text editors and IDEs, C20 Shell Scripting, and all D-level modules}. Quickly browsing and going back when you need more details is fine.

If you emphasize credits (and are in Finland):

> You can earn credits from this as well. One credit comes from the basics (A), related scientific tools (B), and Linux basics part (C). One credit comes come from high-performance computing (D) and intermediate programming strategies (E). If you are a student at Aalto University, the course code is CS-E400401, can be directly added to study plans, and you can begin working at any time. If you are in Finland but not at Aalto, you may enroll and gain credits through the *FITech program*.

What content should be added here?

This site isn't about basic programming courses or the kinds of things you would teach in your courses. It also isn't about writing our own new material - almost everything can be found online, somewhere. We want to make that easy to find. It also doesn't replace a dedicated course on these topics: this is more informal and just a starting point.

It is likely that you will know of or find some **better videos/reading material than what we have currently linked**. Please send that to us! We have two categories, short video intro intros (hopefully with examples) and then longer reading, about an hour or so.

There may also be modules (topic + description + video + reading) which could be added. We will try to have some editorial judgment to keep the "main list" short, but we have an "extras" area for each level.

To submit a proposal, please use the GitHub repository if possible. You can read more about directly modifying the source with pull requests on the contributing page.

4.6.6 CodeRefinery

CodeRefinery is a course on tools needed to do efficient research software development. In-person and online courses are occasionally offered, however, all material + videos are available online. This page collects this material so that you can study on your own.

This page contains an index to all material in one place, in the order it is actually presented, and updated with the current "best" material as we produce new versions of videos / material.

How to use this material

You may go through this at your own page: written and video material are roughly the same and compliment each other; use one or the other or both in whatever order suits your styles.

- **Written lesson material** could be used without the videos.
- **Videos** are self-sufficient for an overview but to do examples you also want to open the written material. They are portrait-mode so that you can adjust your screen to have half of it for you.

- **Q&A** are the live Q&A/notes asked by workshop attendees and answered during the workshop, and are optional (could be used for advanced study).

In the Hands-on Scientific Computing scheme, most of this material is the E-level, with the git-intro being C-level. This page is outside of the main Hands-on SciComp flow and there are no credits directly offered for this page.

Git introduction

The git version control system, from the very basics. How to use it well for your own projects. Topics include: why version control, git, terminology, branches, merging, conflict resolution, inspecting history, undoing things, staging area, practical advice.

- Overall workshop intro
- Lesson
- Video day 1
- Video day 2
- Day 1 Q&A
- Day 2 Q&A

Git collaborative

How to use Git with multiple people. Topics include: collaboration workflows (centralized and distributed), remotes, pushing/pulling, pull requests (merge requests), Github, more on branching and merging, conventions when contributing to other projects.

- Lesson
- Video
- Q&A

Reproducible research and FAIR data

It is easy to do things once, but it's important to be able to do them many times, or for others to be able to do them. Topics include: motivation, organization of files in projects, environments (virtualenv, conda) and recording dependencies, automating computational steps, sharing code and data.

- Lesson
- Video
- Q&A

Social coding and open software

Eventually, you need to use the code or results that someone else has made - or need for others to be able to use your creations! Topics include: why we share, benefits to you, barriers to sharing, encouraging reuse, licenses, citation of software.

- Lesson
- Video
- Q&A

Jupyter

Jupyter is a system for interactive computing. Topics include: why notebooks, best practices, tips and tricks, the Jupyter ecosystem, basics of Jupyter, notebooks and version control, sharing notebooks.

- Lesson
- Video
- Q&A

Documentation

Documentation is often the difference between reusable (or usable by yourself in six months) and not. We go over various ways to make documentation much more enjoyable. Topics include: types of documentation, popular tools, in-code documentation, readme files, the Sphinx documentation generator, hosting docs on ReadTheDocs or Github Pages.

- Lesson
- Video
- Q&A

Software testing

Automatic testing is one of the cornerstones of modern software development and without it, you often end up sending more and more time fixing old bugs rather than doing new things. Here, we the concepts and simple strategies for getting started. Topics include: motivation, relevance to scientific accuracy, pytest, local testing, automated testing (Github Actions), test design.

- Lesson
- Video
- Q&A

Modular code development

When you can mix-and-match and reuse code, your productivity goes way up, and that is enabled by modularity. Here, we give a basic intro to the concept and how to do so. Topics include: what is modularity, why, functions, modules, state and pure functions, unit test, command line interface.

- Lesson
- Video
- Q&A

Concluding remarks and where to go from here

- Lesson
- Video
- Q&A

Other

- Expanded video Q&A from the May 2021 workshop

Source material

Source material from past workshops (in general newer is probably better):

- All CodeRefinery lessons (includes a few minor ones not in the main workshop flow).
- May 2021
  - Workshop page
  - YouTube playlist
  - Q&A
- May 2020
  - Workshop page
  - YouTube playlist

See also

Subscribe to the CodeRefinery newsletter to be updated of when workshops are opened.

4.6.7 HPC Kickstart

This page contains a virtual high performance computing (HPC, or more precisely, cluster computing) kickstart course. It is not part of the main Hands-on Scientific Computing flow, but is an expanded version of the "D" level material.

This page currently contains an online course from Aalto University (Aalto Scientific Computing), so the exact examples may not work on other clusters, but the theory and concepts *will* - you need to combine this outline with documentation from your own site.

In the future, this page will be adjusted to the best topics in the best order from all courses combined, which means various material may be mixed-and-matched so that the transitions are not perfect, but it will still have the best effect overall.

Introductory material

These can be used in whatever order suits you, or you can watch the intro and then go on.

- Day 1 introduction (Video, Lecture)
- HPC theory crash course: some background about high-performance and cluster computing, not strictly necessary to move on to the other material (and could even be watched at the end) (Video, Slides)
- How to ask for help with supercomputers (Video, Slides)
- Your future in scientific computing. (Video, Outline)

Main tutorials

"How to connect and use software/data" track:

- Connecting to the cluster (Video Reading **'Q&A <>'__**)
  - Accounts, ssh, ssh keys, different operating systems, Jupyter, remote desktop environments
- Data storage * Video, Reading, **'Q&A <>'__**)
  - About storage, different storage locations and properties, quotas, access on other computers, remote access
- Applications on the cluster (Video, Reading, **'Q&A <>'__**)
  - How to use other software, common applications, singularity containers, requesting new software
- Software modules (Video, Reading, **'Q&A <>'__**)
  - The `module` command, searching for modules, loading modules, module versions, module collections.

"How to actually run stuff" track. This goes into detail about the batch system and accessing resources:

- Interactive jobs (Video, Reading, **'Q&A <>'__**)
  - Scheduling systems, Slurm, requesting resources, running jobs you can see directly.
- Serial jobs (Video, Reading, **'Q&A <>'__**)
  - Jobs that run without your interaction, scripting jobs, checking output, viewing history, cancelling jobs.
- Monitoring jobs (Video, Reading, **'Q&A <>'__**)
  - Checking actual resource usage of jobs (CPU/memory/GPU) while running and after finished, adjusting resource requirements, reducing resource wastage.
- Parallel jobs (Video, Reading, **'Q&A <>'__**)
  - Types of parallelism, shared memory (OpenMP), message passing (MPI), multiprocessing, how to run each of them, monitoring performance (doesn't cover writing new programs that can do this).
- Array jobs (Video, Reading, **'Q&A <>'__**)
  - What is an array job, doing the same thing many times, serial job → array job, various tips and examples.
- GPU jobs (Video, Reading, **'Q&A <>'__**)
  - GPU programs, machine learning frameworks, compiling CUDA code, requesting a GPU, monitoring efficiency, common efficiency traps.

Special topics

These special topics can be used in whatever order suits you, if they are relevant to your interests.

- Scientific computing workflows: different ways of actually using computing resources. Recommended to put the cluster into perspective with other types of needs. (Video, Reading, **'Q&A <>'__**)
- Currently available resources at CSC, Finland: The above material is mostly abut what you can find at one university on a cluster (though even bigger clusters use the same interface). This talks about other resources available at a national computing center (other countries will be somewhat similar). (Video, Reading, **'Q&A <>'__**)
- Cluster etiquette: We learned what you can do, but what *should* you do to not annoy others on the cluster? See more in Research Software Hour (Video)
- "How to tame the cluster", mostly the same material as this whole course, compressed into one hour, with a complete example worked out. (Video)

See also

- Full playlist of June 2021 Aalto kickstart course and the course page.

4.6.8 Similar resources

Hands-on Scientific Computing is not unique, similar material can be found all around the internet. In fact, this is what we direct you to. HoSC organizes it into one place for you.

Similar large courses/workshops

- Software Carpentry is basic-level material, focused on programming, basic unix commands, and version control.
- CodeRefinery is an intermediate level extension to people who do programming: more version control and basic software development practices that researchers need to know.
  - *A virtual CodeRefinery course* is hosted on this site - lessons, videos, notes all linked together.
- *Virtual HPC Kickstart* course hosted on this site (focused on Aalto University but useful to anyone).