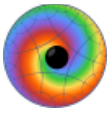
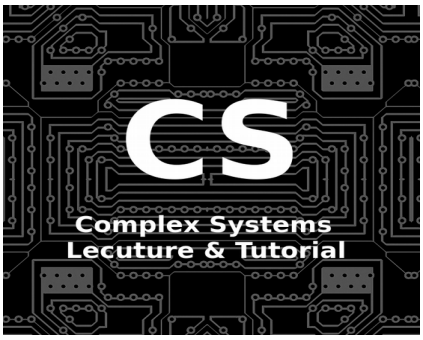


Tutorial on Computational Methods

for studying the Physics of Complex Systems

Prof. Dr. Haye Hinrichsen
Lehrstuhl für Theoretische Physik III
Universität Würzburg



This tutorial gives a short practical introduction into computational methods and numerical techniques which are useful for the study of complex physical systems.

Integrated Development Environment (IDE)

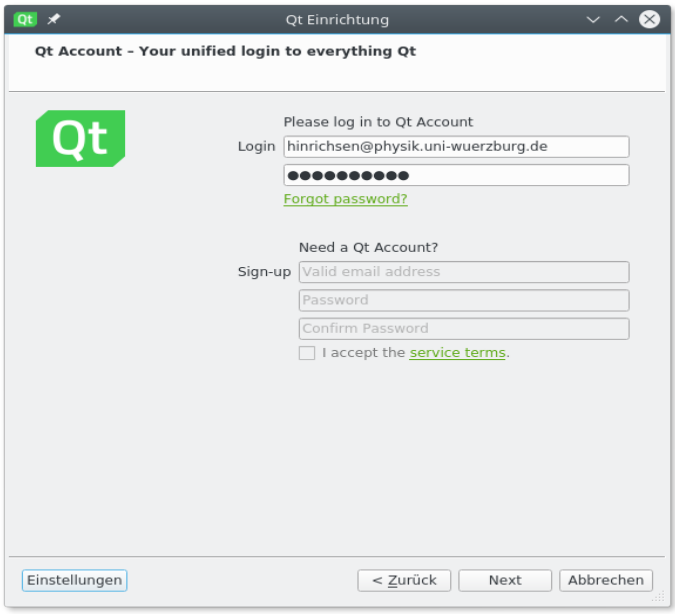
In order to write programs efficiently, we recommend to use an *integrated development environment* (IDE). A large variety of IDEs is available for C++, Java and other languages. In this course we will use **Qt** for several reasons:

- Qt is a professional well-supported environment but it is also available for free (for open-source projects licensed under GPLv3)
- Qt is designed as a multiplatform environment, available on Linux, Windows, Mac and also suitable for mobile devices
- Unlike other IDEs, which produce a lot of code ("overhead") when opening a new project, Qt produces only two files: a project file (*.pro) and a main file (main.cpp). Therefore, Qt is particularly user-friendly for beginners.
- Qt come with an API (Application Programming Interface, a collection of C++ classes) which is very powerful and easy to use.
- Qt can be installed locally without requiring superuser privileges.

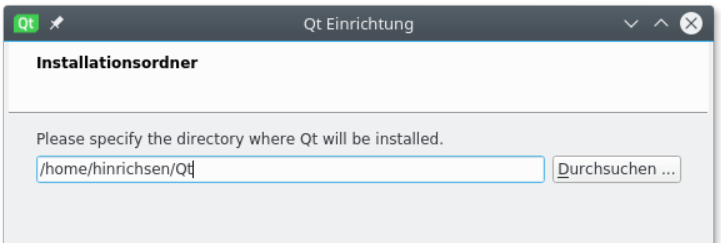
To use Qt you need a computer or laptop with 1-2 GB of free disk space. To get started, download the open-source version of Qt from:

https://www.qt.io/download-qt-installer

For example, on a Linux computer you'll get a file named *qt-unified-linux-x64-3.0.4-online.run* or similar. Right-click the file and change the rights to make it executable. Then run the file by clicking it. The installer window will open, which looks like this:



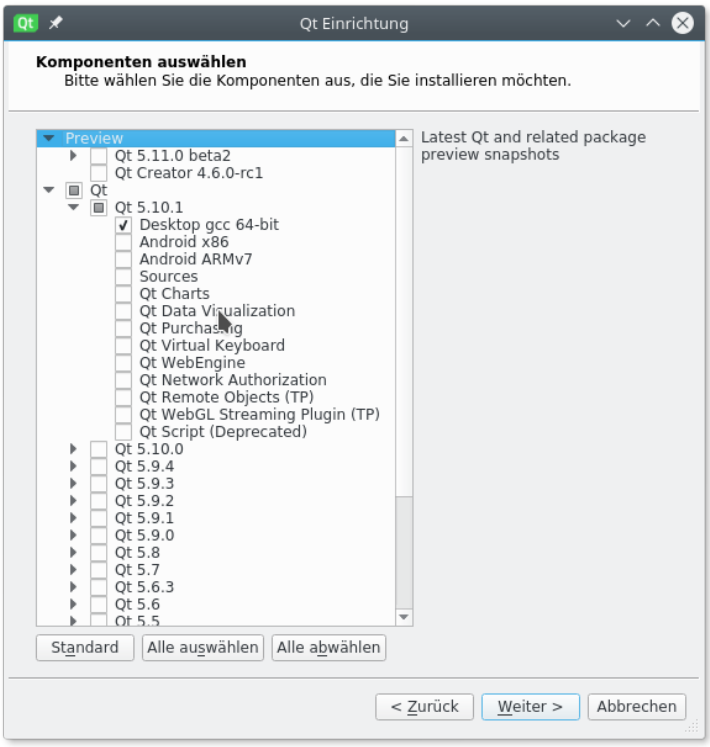
The first thing is to open a new Qt account, which you have to confirm by E-Mail as usual. Then, after receiving some meta information, you will be asked for the folder in which Qt will be installed:



Next you have to select the components to be installed. For our purposes we just need the newest Desktop version and the IDE itself, which is named QtCreator and included by default.

Do not select the "beta" and "rc" versions at the top of the list. These versions are the current development versions which are not yet stable.

This is what it looks like:

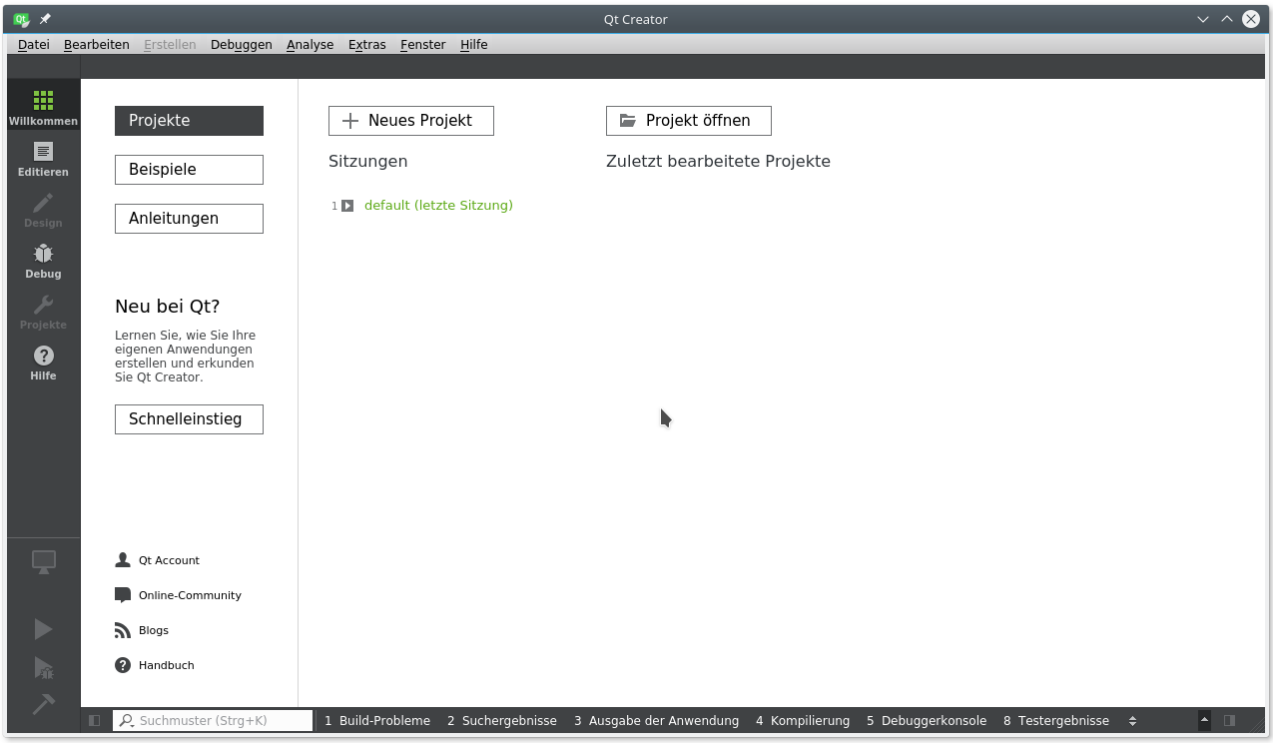


Then press "Next", accept the license and install the software.

If you are done start the QtCreator. If there is no symbol on the desktop, the QtCreator is located in the folder

...../Qt/Tools/QtCreator/bin/qtcreator

This is the QtCreator:



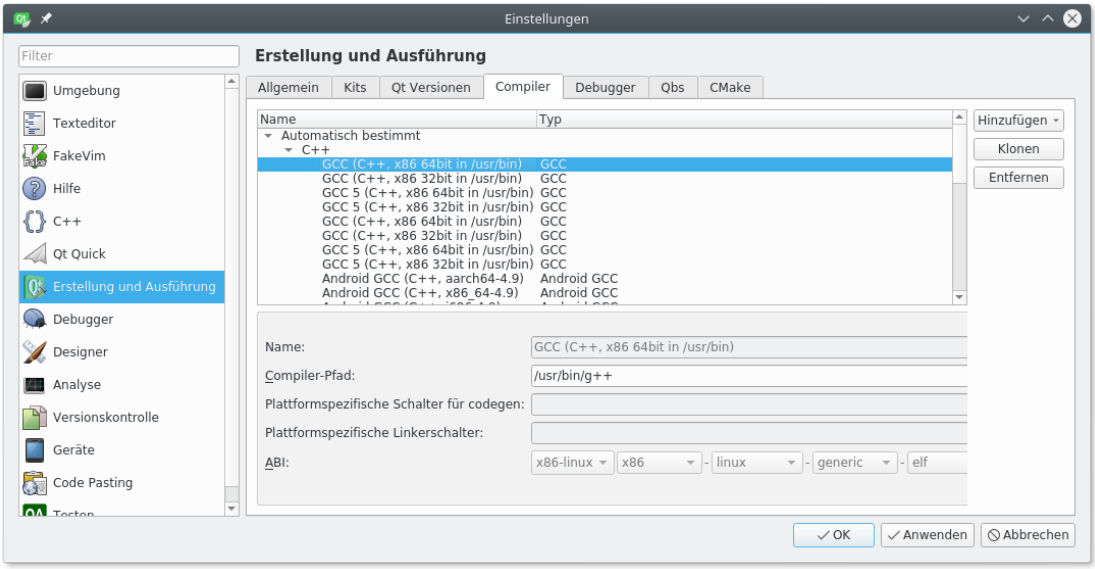
First remark:
At any time it is possible to add/remove/update components of Qt using the "Maintenance tool" which is located in the top level of the Qt installation directory. It looks exactly like the installer:

Second remark:
The QtCreator is only an IDE, not a programming language. It can handle virtually any programming language. This means that it does not come with an in-built compiler, rather it

looks for the compilers that are installed on the system. In this lecture we are primarily interested in C++. This means that we have to install a C++-compiler:

- On Linux install the packages like gcc, g++ (sudo apt-get install ...)
- On Windows the simplest method is to install the open-source MINGW compiler using the Qt-Maintenance tool which is shipped together with Qt. Alternatively you may install Microsoft Visual Studio Community 2017 (this is an IDE in itself).
- On Max OS-X install XCode.

Having installed the compiler, we have to check whether it is recognized by the QtCreator. To this end start the QtCreator, select 'Preferences', choose 'Build and Run' in the left toolbar, and choose the tab "Compiler"



This tab contains a list of all automatically determined compilers. For Linux, there should be some version of GCC in the list. If there is no compiler in the list, there is a problem with the installation of the compiler.

Next we have to verify the 'Kits'. A 'Kit' is a consistent set containing a compiler and the Qt API for a specific platform. In the following example we have a kit for the Desktop and another kit for Android cellphones:

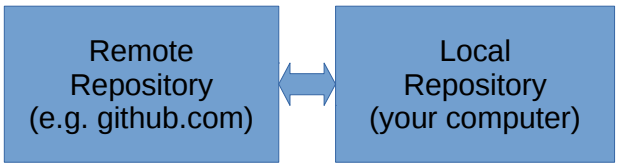
If there are yellow warning signs or red stop signs hover over them with the mouse to see a popup explanation. Your Qt installation is ready if and only if there are no such signs.

If you have any problems with the installation please do not hesitate to come over to my office.

GIT Version Management

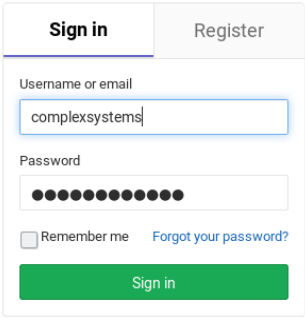
Professional programming requires the use of a version management system. Here we give an introduction to a widely used open-source system called GIT.

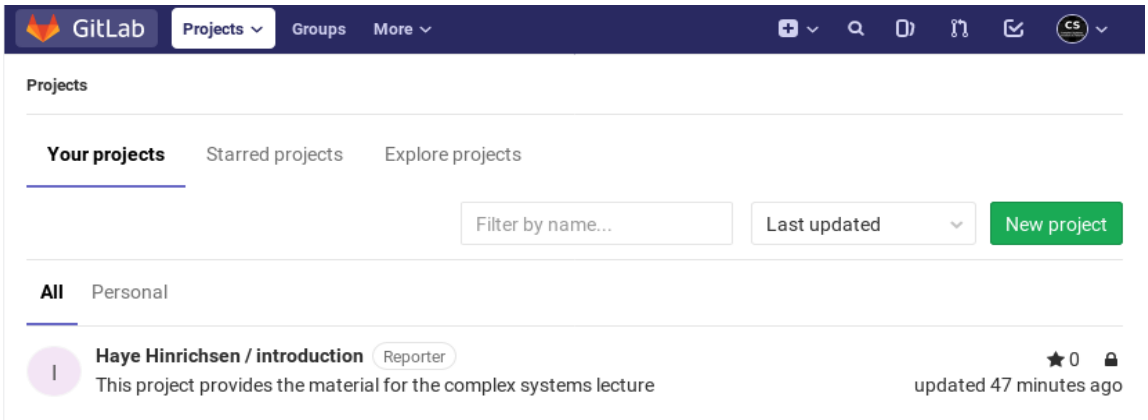
Git works with any text-only based files. Starting point are so-called **repositories** which are basically collections of files. Typically one has for each project a **local repository** which is just a folder on the local file system containing the files to be monitored. In most cases, this local repository communicates with a **remote repository** on a public server. Free servers include github.com and gitlab.com, but also our physics departments runs a server (git.physik.uni-wuerzburg.de, but only for registered invited users). The purpose of the remote repository is to give several users access to the same project and to collaborate.



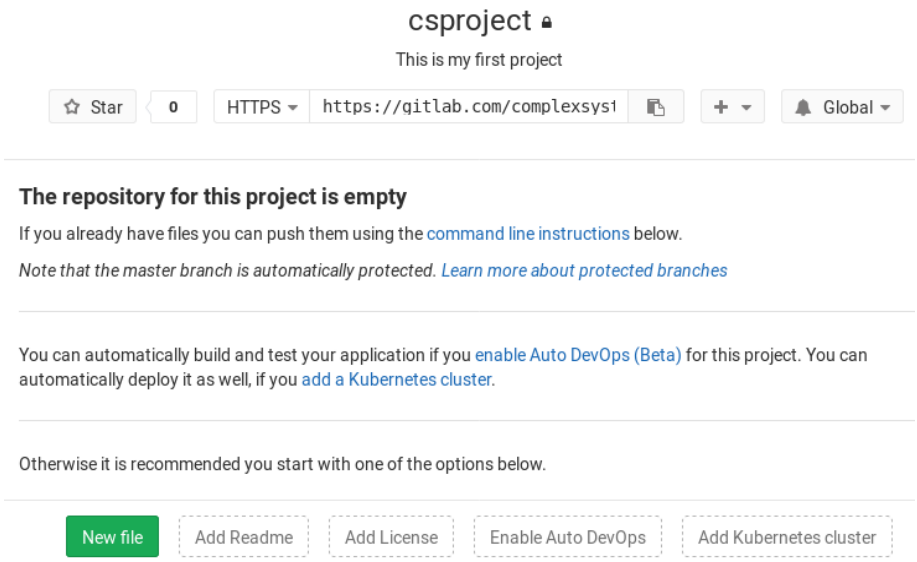
Creating a project on the remote server:

For our course we have created a free account on gitlab.com with the username 'complexsystems' and a password to be announced in the tutorial. Go to gitlab.com and sign in. Then you will see a list of the projects (=remote repositories) in this account:

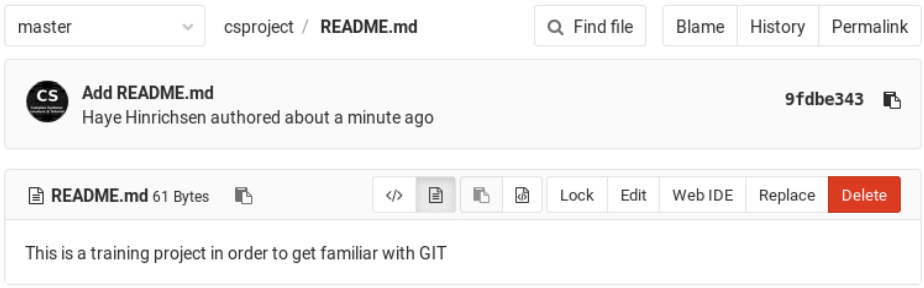




To create a new project press the + button in the upper toolbar, select "new project" and specify the project name. In the following example we create a project named "csproject". Keep the project at the 'private' visibility level so that only the participants of our lecture have access to it. Finally, press 'Create project'. The system responds with:



It is a good practice to first press "Add Readme" and to create a little readme file in the remote repository explaining the purpose of the project. Type some lines and finally press "Commit Changes". What we'll see is the following:



What we see is the following: In the main branch (branches will be explained later) called "master" of the project 'csproject' there is now one file with 61 characters called 'README.md'. There has been one 'commit' with the subject "Add README.md" and the hash 9fdbe343 (this is a long hexadecimal number identifying the commit). In the lowest line the content of the readme file is displayed (the extension *.md stands for 'markdown' which is like *.txt with some additional formatting options).

So far all this took place on the remote server at gitlab.com. Next we want to copy ('clone') the project to our own computer. More precisely, we want to create a local repository on our own computer with the same content .

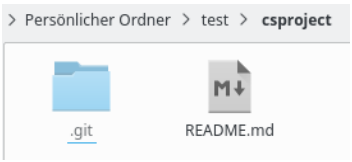
To copy the files open a terminal (a console). In the following we assume that you have already installed git on your computer (e.g. on Linux by 'sudo apt-get install git'). To see whether git is installed simply type 'git' and you should get a long list of help.

Now go to a directory where you want to create the local repository. Then type:

git clone https://gitlab.com/complexsystems/csproject.git

```
hinrichsen@laptop ~ $ cd test
hinrichsen@laptop ~/test $ git clone https://gitlab.com/complexsystems/csproject.git
Klone nach 'csproject' ...
Username for 'https://gitlab.com': complexsystems
Password for 'https://complexsystems@gitlab.com':
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Entpacke Objekte: 100% (3/3), Fertig.
Prüfe Konnektivität ... Fertig.
hinrichsen@laptop ~/test $ ^C
hinrichsen@laptop ~/test $
```

This should create a folder named complexsystems containing the readme file:



Enabling the option 'show hidden files' you will also see a hidden folder named .git with a dot in front. This folder contains the whole data for the version management, so to say the whole history of the project in a highly compressed and very efficient format. Fortunately, we don't have to care about it. Switch 'show hidden files' off and forget about it.

Creating a project on the local machine:

Conversely it is also possible to create a project first on the local computer and then to transfer ('push') it to the remote server. To this end let us first create a folder on the local machine, for example 'secondproject'. In this folder we create a file README.txt fill it with some lines of text. At this point it is just a folder with a txt file. Git is not yet involved.

To connect it with git, open a terminal (console) and enter the folder where you created README.txt. First initialize the git system by typing

git init

Next, inform git about your identity by typing

git config --global user.email "hinrichsen@physik.uni-wuerzburg.de"

git config --global user.name "Haye Hinrichsen"

replacing my name and email with yours – this has to be done only once. The next thing to do is to register the readme file for surveillance by adding it to what git calls the "stage" or "staging area". This is done by typing

git add README.txt

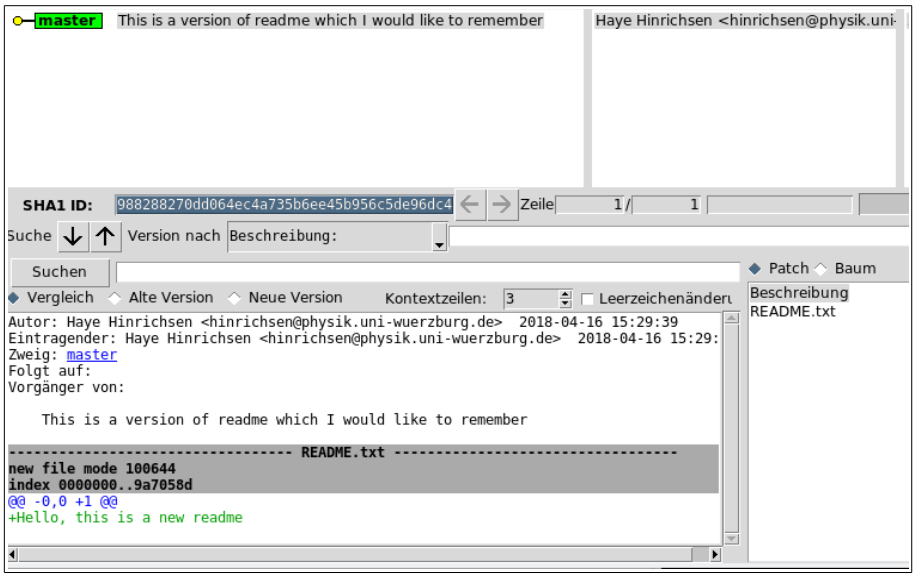
If you simply want to register all your files in that folder, it is also possible to type

git add .

The next step is to mark the present version of README.txt as a 'commit' which you would like to be able to restore in the future (usually a 'commit' marks a certain important point of progress in the development of a project). This can be done by

git commit -m "This is a version of readme which I would like to remember"

Now we have one commit on our local machine. On many systems there is a visual frontend called **gitk** which allows you to view the commits (if not install it). Typing **gitk** yields



In the upper line you can see that there is a single commit in the present branch called "master". Below you find the ID of the commit (marked in blue) which is a long and virtually unique hexadecimal number. Further down details about the commit are given, e.g. the author, the file, the content and so on. Note that **gitk** is only a passive viewer, you cannot use it to influence or control the git system.

So far the new project lives only on the local machine. Now we would like to submit it to the remote server. To this end we have to inform git about the location of the server. This can be done by typing

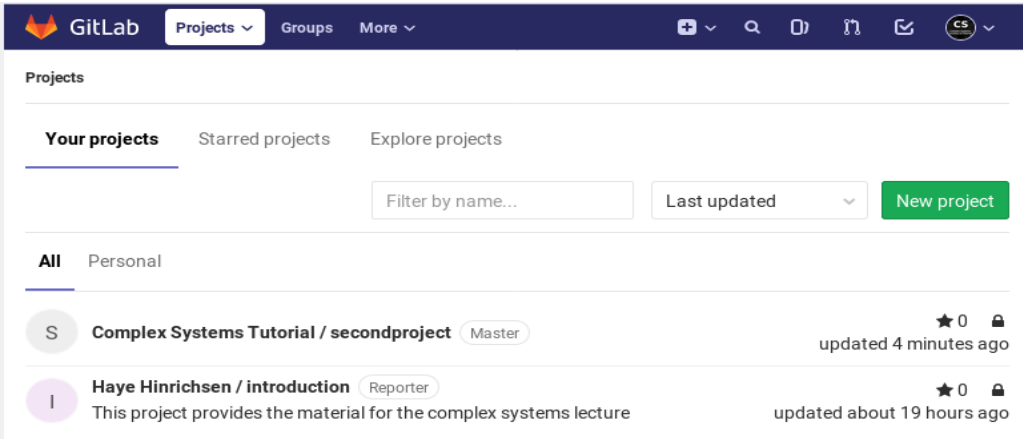
git remote add origin https://gitlab.com/complexsystems/secondproject.git

With this command the gitlab.com server is declared to be the "origin" of the project. Note that the path includes the account (complexsystems) and the project name (secondproject). If you want to know whether and to which remote server the project is connected, simply type **git remote -v** and you will see the path of the remote server.

The 'git remote add' only specifies the remote server but it does not yet upload the project. This can only be done by 'pushing' the project via

git push origin master

which means that the present branch (master) is uploaded to the master branch of the remote server (origin master). Now, if we open the web-based frontend on gitlab.com, we can see the "secondproject":



How git works

Git has essentially three levels: The **working directory**, which is just the directory of the project containing your files, a **stage** keeping track of the files of which git is supposed to

monitor, and a unidirectionally linked **list of commits**, each of them carrying a unique hexadecimal ID. A pointer called HEAD points to the latest commit.