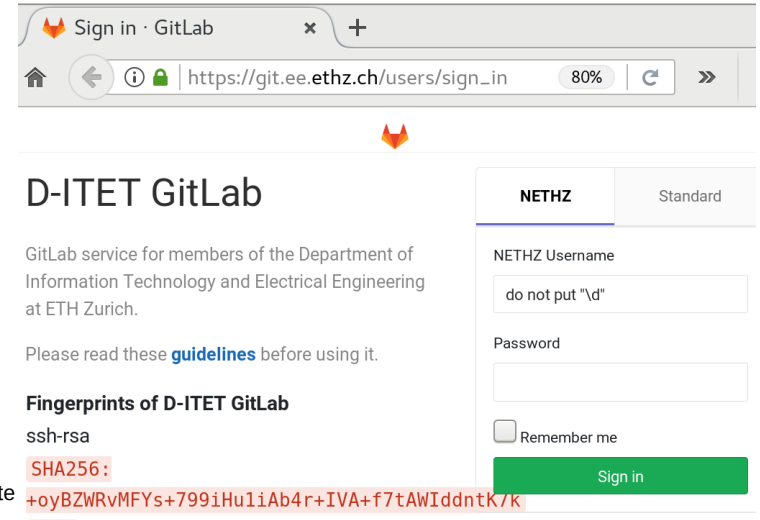
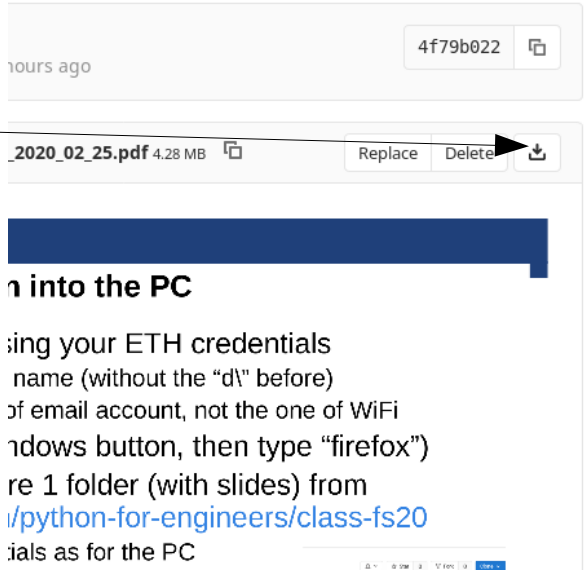
While waiting: login into the PC

- Login into the PC using your ETH credentials
  - User: the ETHZ short name (without the “d\” before)
  - Password: the same of email account, not the one of WiFi
- Open a browser (windows button, then type “firefox”)
- Download the Lecture 1 slides from https://git.ee.ethz.ch/python-for-engineers/class-fs20
  - use the same credentials as for the PC
  - Slides are in the folder `Lecture_01_introduction`
  - to download click on the icon on the right





ETH zürich

Python for Engineers

- get productive in the classroom, in the lab and at home

Lecture 1: Introduction

Luca Alloatti, luca.alloatti@ief.ee.ethz.ch
Raphael Schwanninger, raphael.schwanninger@ief.ee.ethz.ch
Thomas Kramer, thomas.kramer@ief.ee.ethz.ch
(Marco Eppenberger, marco.eppenberger@ief.ethz.ch)

Introduction to the course

- This is the fourth time that this lecture is offered. Python was not taught before at D-ITET.
- Agenda of the course:

25.02.2020: Introduction (today)
03.03.2020: Python Basics 1
10.03.2020: Python Basics 2
17.03.2020: Python Basics 3
24.03.2020: Object Oriented Programming
31.03.2020: Software Licenses
07.04.2020: Graphical User Interfaces
14.04.2020: – Easter Break –
21.04.2020: Scientific Tools 1
28.04.2020: Plotting
05.05.2020: Scientific Tools 2
12.05.2020: Laboratory Automation
19.05.2020: Profiling and Performance
26.05.2020: you decide (multiple choice)

You can find a PDF with all course information in the Git repo.

PC vs. own laptop

- All PCs run the Debian operating system. They are managed by the IT Support Group ISG.EE. It is not possible to have root access.

- This course will support only Debian and the provided desktop PCs. However, you are encouraged to use (at own risk) your own laptop and other operating systems if you wish. Software needed:
  - jupyter-notebook
  - python 3.7
  - git

Requirements for receiving credits

- You must have seriously worked on all provided exercises.
  - Except the ones marked with “extra”.
- You must have uploaded all your work in your Git repo fork by the end of the semester (we will see later how you can do this).
- As with any P&S, there is no grading but only a “pass” and “fail” mark.

Today

- Introduction
  - GNU/Linux ‘Debian’
  - GNOME Desktop
  - Terminal
  - Git (source-code managment)
  - A 'hello world' program in Python

Why Python

- Together with C, one of the most popular programming languages
- Interpreted language (instructions executed step-by-step, as opposed to previously compiling the code into a binary)
- The syntax emphasises code readability
- Supports multiple paradigms, such as:
  - Imperative (like C – program consists of statements that somehow mutate data)
  - Functional (focus on evaluation of mathematical functions, avoid mutable data)
  - Object oriented
- Very large and extensive set of library or modules
- Open source, permissive license, gratis
- Python + its ecosystem (numpy,…) could be the only tool you need for most of your programming experience at ETH (replacing matlab, java, plotting, data analysis, lab automation)
- Can be used to write portable code, but it is also possible to write code which is specific to the operating-system

Some close-by groups using Python

- Sensirion.com is an example close to us (Staefa) using massively python
  - It also sponsors the Swiss python-summit.ch (Rapperswil)
- Python hackathon at the 2017 European Conference on Optical Communications (ECOC) and at the 2018 Optical Fiber Conference (OFC) :

  http://python4photonics.gitlab.io/hackathon/
- Our group, and many others at ETH
- ...

SENSIRION
THE SENSOR COMPANY

History of Python

- Started in 1989 Guido van Rossum (Dutch), first released in 1991
- He was looking in 1989 for a hobby program to keep him busy during Christmas vacations

  "Python is a new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers"
- Name is not related to the snake, but to the British comedy series “Monty Python Flying Circus”

Python versions

- Python 1.0 released in 1994
  - with standard data types, classes with inheritance, functional programming
- Python 2.0 released in 2000
  - added list comprehensions, garbage collection,… , and a more transparent community-based development
- Python 3.0 released in 2008
  - rectify fundamental earlier design flaws, Python “grew up”
  - reduce feature duplication by removing old ways of doing things
  - 5 / 2 is 2.5 and not 2
  - no full backwards compatibility
- Python 3.7 released on June 27th 2018

Short introduction to Debian

- Debian is one of the several GNU/Linux distributions
- It is the distribution with the largest number of packages making it the largest collection of software in the world
- It supports several desktop environment such as KDE or Xfce. In this course we will use the default desktop environment GNOME. Some commands in the following are unique of GNOME.
- A desktop environment is not necessary to run Debian: most commands can be executed from the terminal...

Zoology of GNU/Linux operating systems

https://en.wikipedia.org/wiki/List_of_Linux_distributions

Filesystem Hierarchy Standard

- / (this is the root, everything is inside here, including disks)
- /dev/ contains devices, and e.g. /dev/zero /dev/urandom
- /home/ (on the D61.1 computers it is saved on the network, not on the physical PC)
- /scratch/ (this is saved on the physical PC → remember the name of the PC to come back to the same machine!)
- /mnt/ for mounting temporary a file system, like a USB
- /tmp/ for temporary files, often discarded after reboot

https://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard
http://www.linuxfromscratch.org/

File permissions

```
drwx------  3 alluca alluca-group    27 Dec 31  2016 projects
drwxr-x---  2 alluca alluca-group    10 Sep 17 14:03 Public
drwxrwx---  3 alluca alluca-group    53 Oct  3  2016 .rve
-rw-rw----  1 alluca alluca-group   347 Oct  7  2016 .rvedb
drwx------  2 alluca alluca-group   100 Dec 31  2016 .ssh
-rw-rw----  1 alluca alluca-group   559 Apr  5  2016 .synopsys_dv_prefs.tcl
drwxr-x---  2 alluca alluca-group    10 Sep 17 14:03 Templates
-rwxrwxrwx  1 alluca alluca-group     7 Sep 24 11:38 test
```

- Type "`ls -la`" → The first letter tells if it is a directory, a link, etc. The following 3 groups of three letters are for "owner", "group", and "others".
  - r = read permission,
  - w = write permission
  - x = execute permission

- chmod +x *fileName* (adds execute permission)
- chmod 777 (adds rwx permission to all)
- rwx = "111" → 7 in decimal
- -wx = "011" → 3 in decimal, etc..

Familiarizing with the terminal command-line

- The file cheatsheet_linux_command_line.pdf contains the most common commands
- In the next slides we will review a few
- Short options of a command can be concatenated, for example `ls -la`

Bash Commands

uname -a	Show system and kernel
head -n1 /etc/issue	Show distribution
mount	Show mounted filesystems
date	Show system date
uptime	Show uptime
whoami	Show your username
man *command*	Show manual for *command*

Bash Shortcuts

CTRL-c	Stop current command
CTRL-z	Sleep program
CTRL-a	Go to start of line
CTRL-e	Go to end of line
CTRL-u	Cut from start of line

Bash Variables (cont)

export *NAME=value*	Set *$NAME* to *value*
$PATH	Executable search path
$HOME	Home directory
$SHELL	Current shell

IO Redirection

cmd < *file*

> Input of *cmd* from *file*

cmd1 <(*cmd2*)

> Output of *cmd2* as file input to*cmd1*

cmd > *file*

> Standard output (stdout) of *cmd* to *file*

cmd > /dev/null

> Discard stdout of *cmd*

cmd >> *file*

Command Lists

cmd1 ; *cmd2*

> Run *cmd1* then *cmd2*

cmd1 && *cmd2*

> Run *cmd2* if *cmd1* is successful

cmd1 || *cmd2*

> Run *cmd2* if *cmd1* is not successful

cmd &

> Run *cmd* in a subshell

Directory Operations

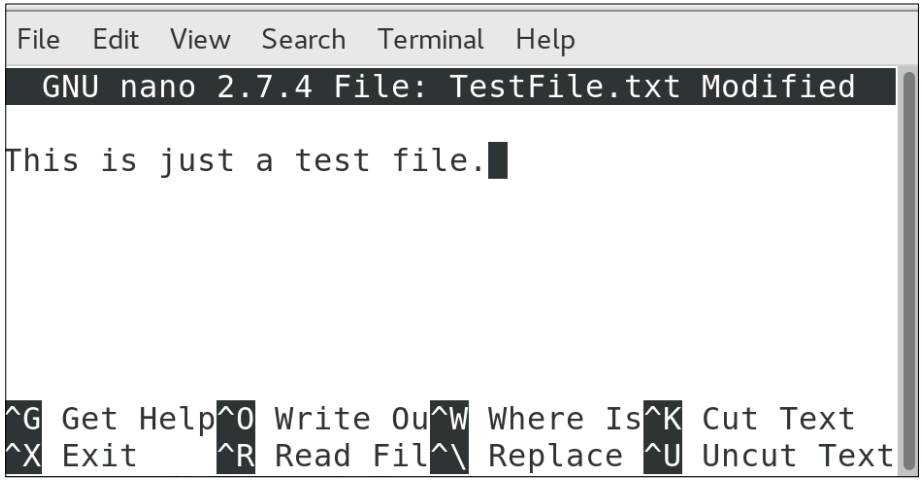
pwd	Show current directory
mkdir *dir*	Make directory *dir*
cd *dir*	Change directory to *dir*
cd ..	Go up a directory
ls	List files

Use the nano text editor

Micro-Exercise (2 min)

- nano is a lightweight text editor which runs in the terminal and is available in default installations. It requires no X Window System (runs also without desktop environment)
- Open a terminal (windows button, then write “terminal”)

- type `nano TestFile.txt` and press enter
- Write some random text
- Copy the text with Ctrl+Shift+c
- Paste the text with Ctrl+Shift+v
- Press Ctrl+x (to exit), then “y”, then Enter, to save

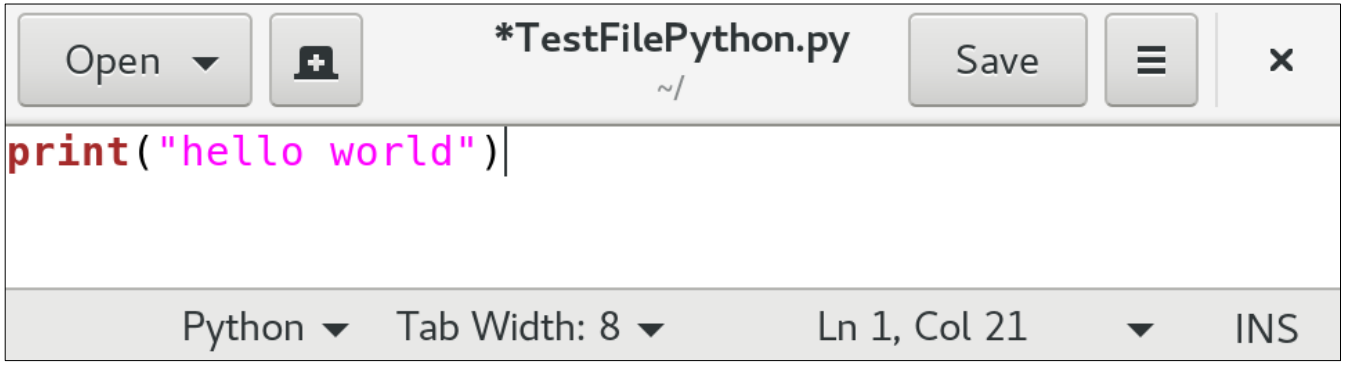


Use the gedit text editor

Micro-Exercise (2 min)

- gedit is a much more powerful editor than nano. It is the GNOME default editor.
- Open a terminal (windows button, then write “terminal”)

- type `gedit TestFilePython.py` and press enter
- if the file name terminates with “.py” gedit will use python syntax colouring



Run some commands

Micro-Exercise (5 min)

- Open a terminal (windows button, then write “terminal”)
- Go through the cheat sheet and test some commands of your choice.
- Suggested actions:
  - list the content of a directory (`ls`)
  - check the file permission (`ls -la`)
  - enter a directory (`cd <directory name>`)
  - go to home directory (`cd`)
  - print the current (=working) directory (`pwd`)
  - create and remove a folder (`mkdir, rm -rf`)
  - move a file (mv)
  - copy a file or folder (`cp` and `cp -R`)
  - search for files containing some pattern (try: `grep -irne “some-pattern”`)
  - close the terminal from command line (`exit`)

Hint: double-check your file operations in a standard file browser window.

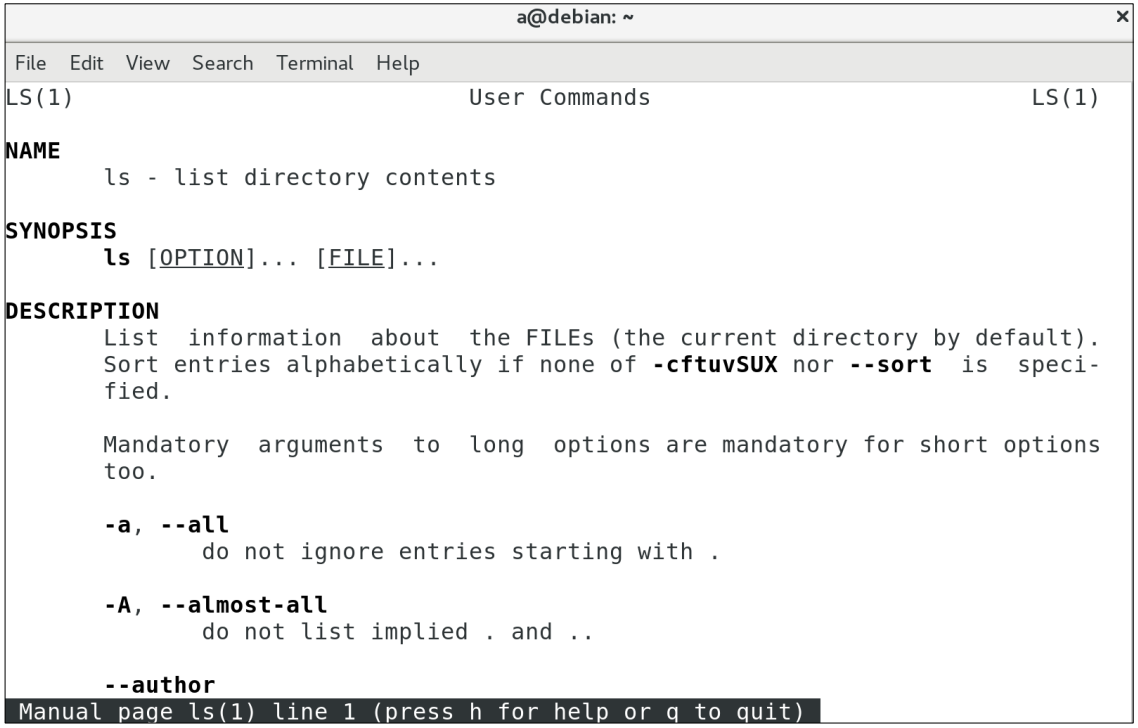
Finding help with “man”

Micro-Exercise (2 min)

- If you have some doubts about any commands, or forgot some of the options, you can consult its manual by typing:

  `man <command>`
- For example, type:

  `man ls`
- To search some specific string: press “/” then write the keyword
- press “h” for more commands
- to quit, press “q”



Find out what the arguments exactly meant for `grep -irne`.

More tricks

Micro-Exercise (10 min)

- Try the following:
  - use the mouse middle-button to copy-paste: highlight some text with the mouse, then move the mouse, for example, in an empty editor, and click the middle-button
  - write “pyt” then press “Tab”: the command will autocomplete
  - use “Tab” to autocomplete names of files or of folders
  - show running processes (`top` , to exit: “q”)
  - kill an application (try `kill` and `xkill`)
  - login remotely into your neighbour’s machine (try `ssh -X <your-ETH-name>@tardis-cXY` , to quit: `exit`)
  - launch a program with and without an “&” at the end (e.g. “`firefox &`”). With the “&” the terminal remains active.
  - stop a (hanging) process (Ctrl + c), for example a `grep` search which takes too long, or cat `/dev/urandom`

Repeat previously-typed commands

Micro-Exercise (2 min)

- Previously-typed commands are saved in a special file (the hidden file .bash_history in the home folder), and can be accessed again by typing "`Ctrl+r`" followed by the first letters of the searched command

- Another way to find previously-typed commands, is by using the arrow-up/down keys (try it out!)

To test it: try for example "`Ctrl+r`" followed by "`ssh -X`". This should find the full ssh command used to access your neighbour's machine in a former micro-exercise. Finally press enter.

Micro-exercise: arrange and navigate through windows and tabs (10 mins)

Micro-Exercise (10 min)

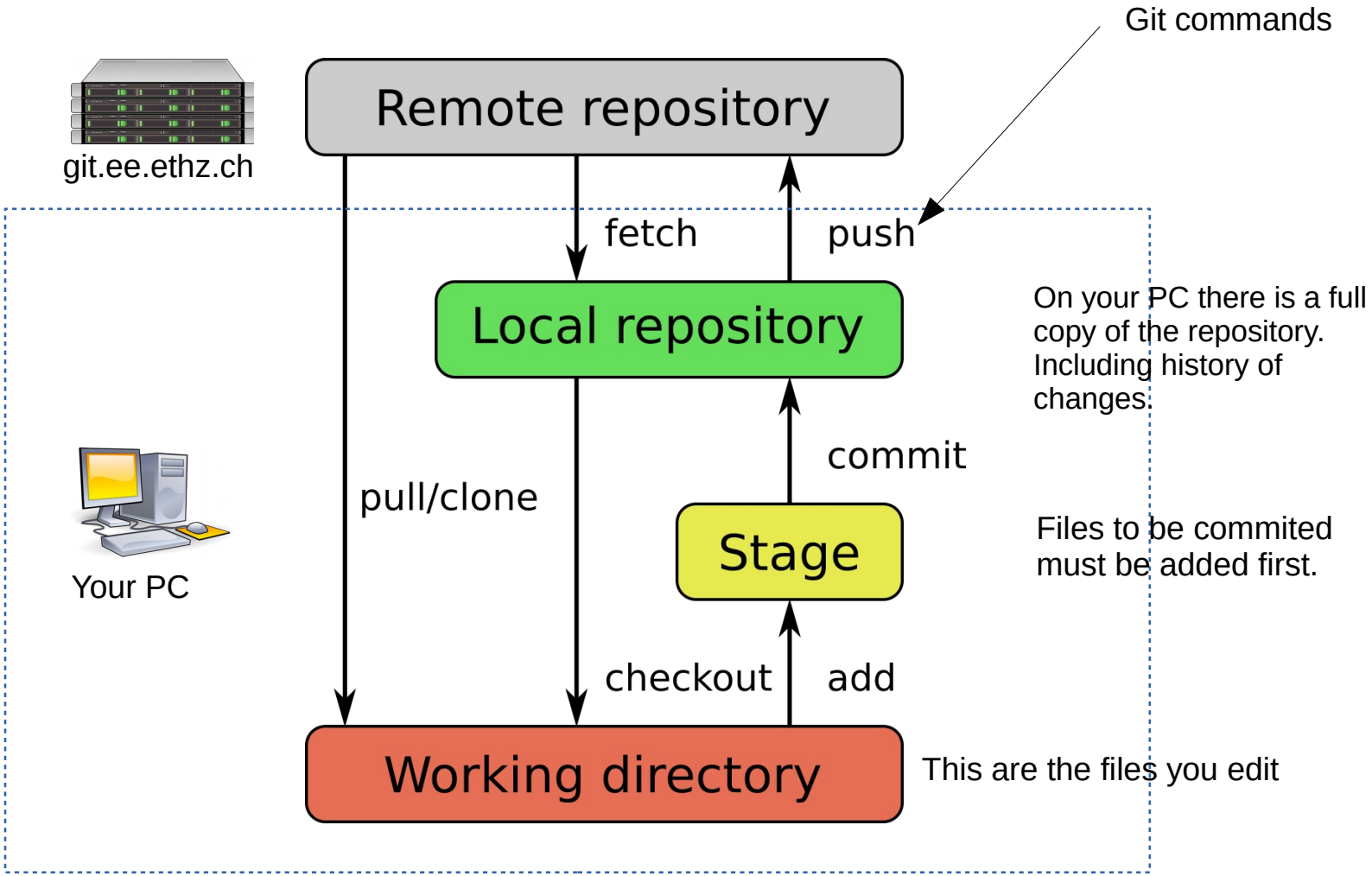
- Experiment with the key combination “window+arrow-key”
  - place a window on the left/right half display, or on full display
- Open a few programs, such as LibreOffice Write, LibreOffice Impress, gedit,…, and experiment with switching between the different windows (“window+tab”)
- Open several instances of the same program such as firefox (“window” key, then write “firefox” then press together “Ctrl + Enter”), then switch between them with the keys “window + `”. Notice that “`” (called *backtick* or *grave accent*) is located just above the “tab” key.

The Git version-control system: Introduction and setup

Introduction to Git

- Git is a version-control system primarily used for tracking changes in *source-code* files and coordinating work within a group of people
- Replaces similar tools *svn (subversion)*, *mercurial*
- It was originally developed in 2005 by Linus Torvalds
- Every user has a full copy of the repository, so even if the main server fails the full history can be recovered
- github and gitlab servers centralize a service which could be more decentralized. There is no similar system which is effectively federated yet, but work is in progress.
- We will learn how to use git "*by example*"

Git – Source Code Management



Setting up Git for the first time

Micro-Exercise (15 min)

- Even if not strictly necessary, this class will use the https://git.ee.ethz.ch gitlab server as a privileged location to store lectures' material and students' solutions
- The remote server needs to make sure that only you can access those files. In principle one could set up a password, but this method has been disabled by the ETH IT administrators. It is necessary to create a private-public key pair, and upload the public key on the remote server
- Detailed instructions are provided at: https://git.ee.ethz.ch/help/ssh/README#generating-a-new-ssh-key-pair
  However, we will run them together in the following slides.

Slides 29 to 43 show how to setup your Git repo and the automated login to be able to participate in the class.

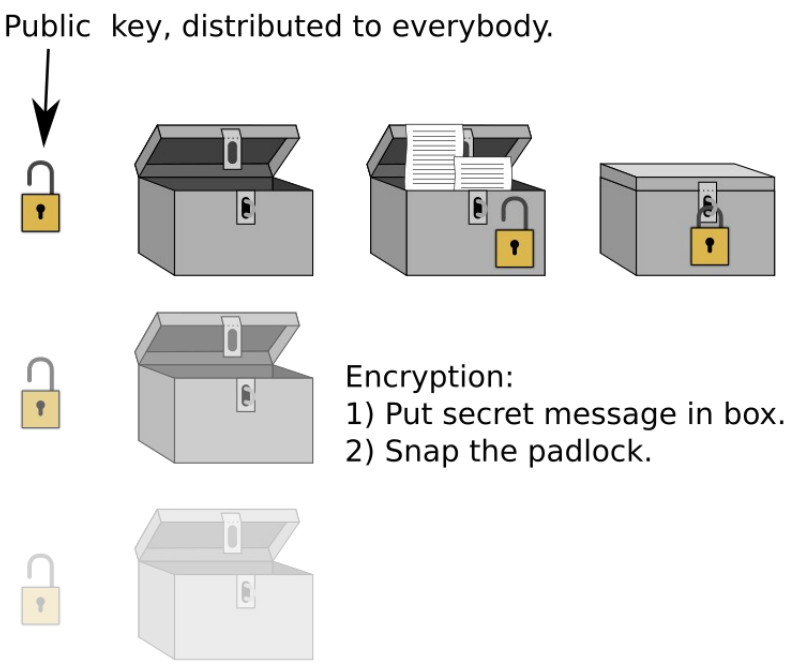
Small reminder: private and public keys

Private key: you do not give it to anybody

Public key: you can give it (as many copies as you wish) to others

There are *infinite* different private-public key pairs

Small reminder: asymmetric encryption



Create a key pair with ssh-keygen

```
File  Edit  View  Search  Terminal  Help
a@debian:~$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/a/.ssh/id_ed25519):

Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/a/.ssh/id_ed25519.
Your public key has been saved in /home/a/.ssh/id_ed25519.pub.
The key fingerprint is:
SHA256:oDuRn41GFxkwCtV8jMlH+VHdY5ke10ZbiCxp9+WGZtk a@debian
The key's randomart image is:
+--[ED25519 256]--+
|  ...+o*o. +.o +*|
|   . .*.=o= + o*O|
|    . .ooo + .oO+|
|     o . ..   *.E|
|    + . S    o . |
|     = =         |
|    o = .        |
|     o           |
|                 |
+----[SHA256]-----+
a@debian:~$
```

Find out with „man ssh-keygen“ what the -t ed25519 option stays for

Press enter a few times to use the default filename and no password.

This is the public key

```
File  Edit  View  Search  Terminal  Help
a@debian:~$ cd .ssh
a@debian:~/.ssh$ cat id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIOJ1J96/oHOIuSHcy+LJoqu28p
PtVEp+oG+CAfpPcYiV a@debian
a@debian:~/.ssh$
```

The public key will be put on the git server later on.

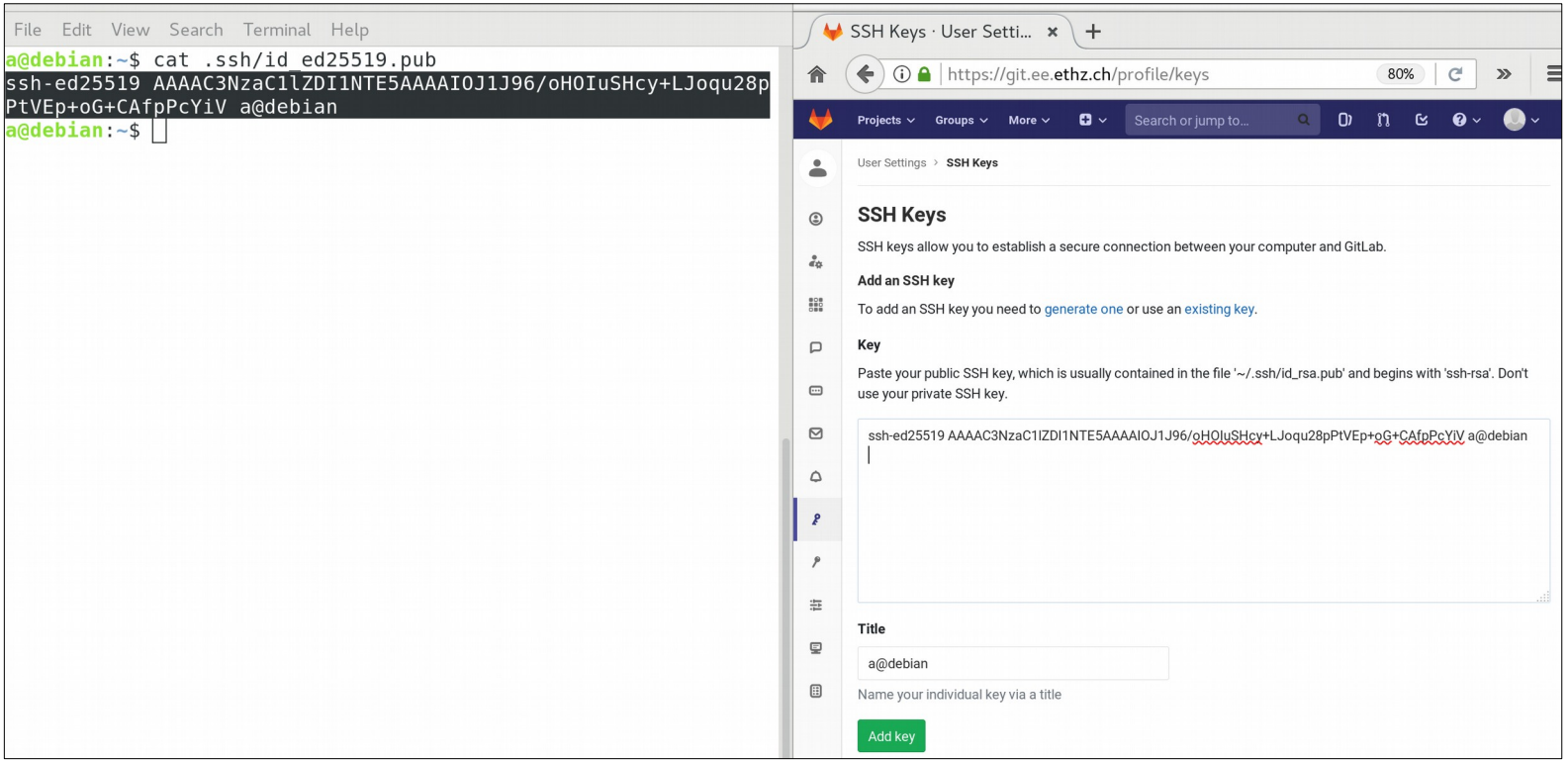
This is the private key (don’t share it!)

File Edit View Search Terminal Help

```
a@debian:~/.ssh$ cat id_ed25519
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW
QyNTUxOQAAACDidSfev6BziLkh3MviyaKrtvKT7VRKfqBvggH6T3GIlQAAAJDx44E+8eOB
PgAAAAtzc2gtZWQyNTUxOQAAACDidSfev6BziLkh3MviyaKrtvKT7VRKfqBvggH6T3GIlQ
AAAEA+uXO6nrCSpbDn9tqkpInKmb8czGndTncdM5FwJxLFS+J1J96/oHOIuSHcy+LJoqu2
8pPtVEp+oG+CAfpPcYiVAAAACGFAZGViaWFuAQIDBAU=
-----END OPENSSH PRIVATE KEY-----
a@debian:~/.ssh$
```

The private key will be used to automatically authenticate you on the git server.

Go to https://git.ee.ethz.ch/profile/keys then upload the public key (highlight text on the left, then paste it on the right by clicking the middle mouse button or copy it with “Ctrl+Shift+c”). Finally click “Add key”.

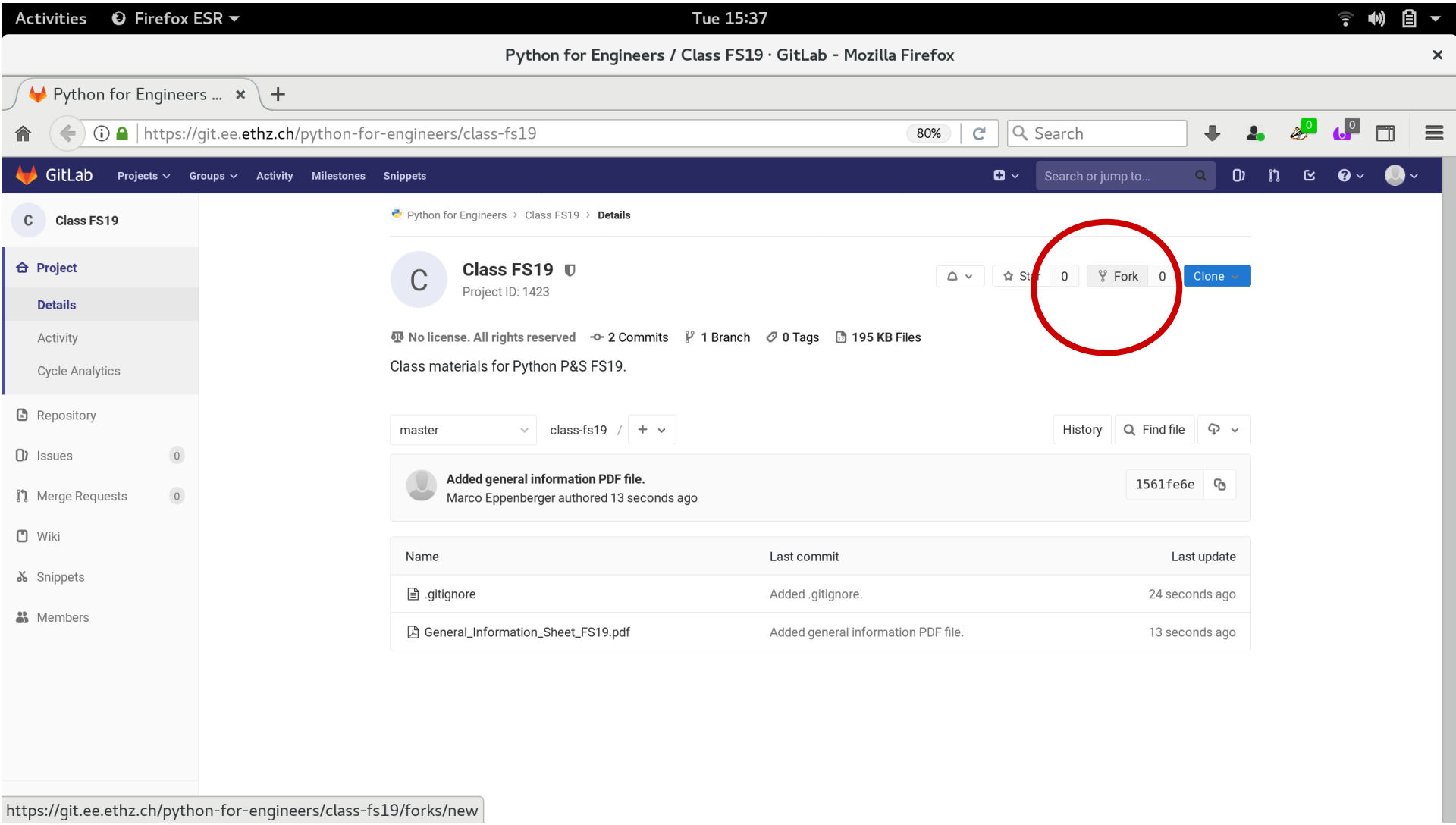


In the future, when interacting from the terminal with the remote server, your computer will *automatically* verify the private-key pair to make sure it is you.

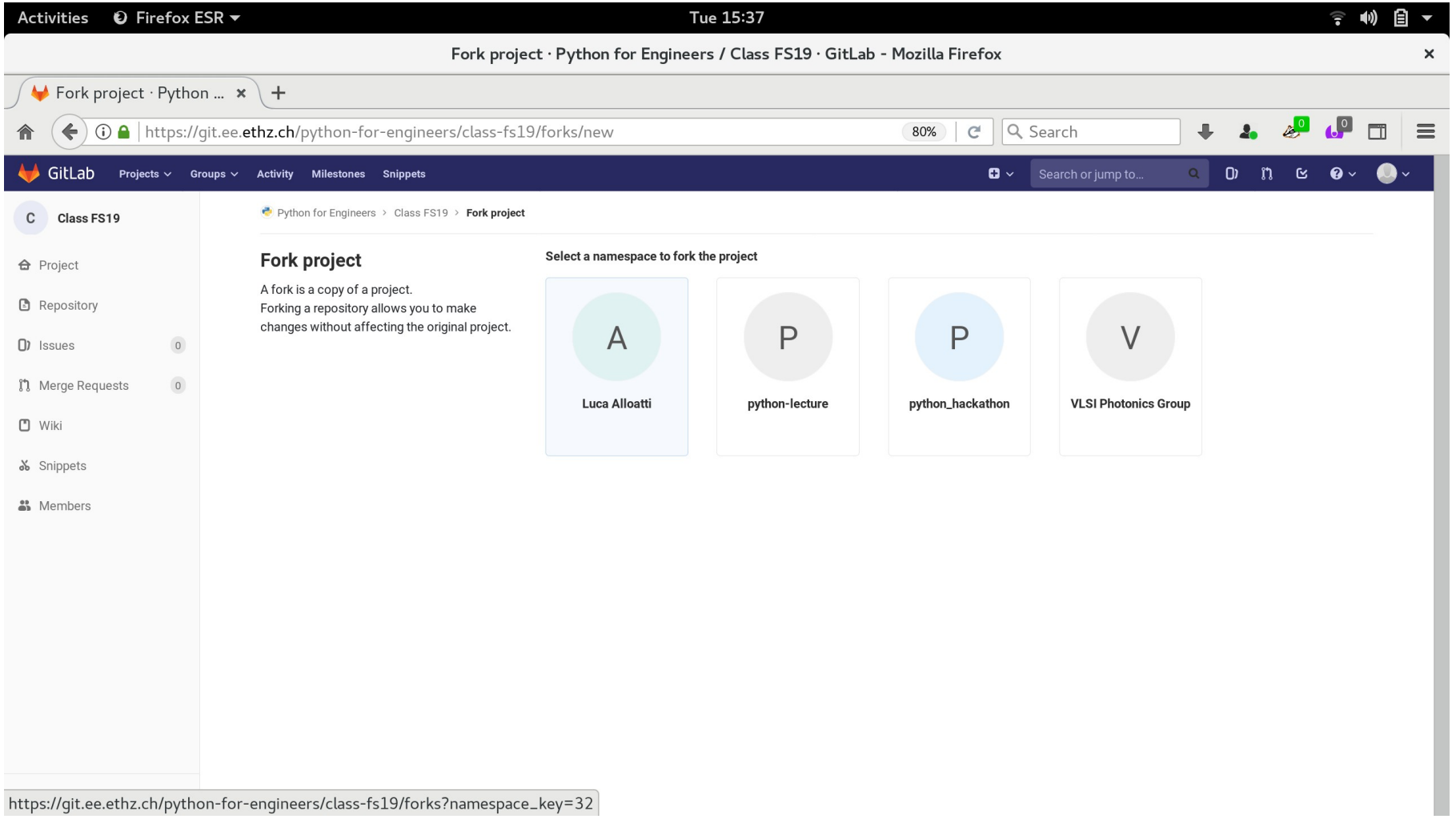
Fork the project we created for you, and give us access to it

- go to https://git.ee.ethz.ch/python-for-engineers/class-fs20
- then click on Fork: a “copy” of the class project is now accessible as one of your projects in your namespace
- In a terminal write:
  - `cd` (this will bring you home)
  - `git clone git@git.ee.ethz.ch:<your-ETH-short-name>/class-fs20.git` (this will clone your fork into your home folder)
  - `cd class-fs20`
  - `git config --global user.name "my name"`
  - `git config --global user.email "MyAddress@student.ethz.ch"`
  - `git remote add upstream git@git.ee.ethz.ch:python-for-engineers/class-fs20.git` (all in one line!)
  - On the web interface, add luca.alloatti@ief.ee.ethz.ch, thomas.kramer@ief.ee.ethz.ch and raphael.schwanninger@ief.ee.ethz.ch as group member, selecting Reporter as access role:
- **Screenshots of each steps are given below**

* go to https://git.ee.ethz.ch/python-for-engineers/class-fs20
* then click on Fork



Select <YourName> as “namespace” for the fork (this step might not be present)



Clone the remote repository in your home with:

`git clone git@git.ee.ethz.ch:<your-ETH-short-name>/class-fs20.git`

a@debian: ~

File Edit View Search Terminal Help

```
a@debian:~/.ssh$ cd
a@debian:~$ git clone git@git.ee.ethz.ch:alluca/class-fs19.git
Cloning into 'class-fs19'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 6 (delta 0)
Receiving objects: 100% (6/6), 90.68 KiB | 0 bytes/s, done.
a@debian:~$ cat class-fs19/.git/config # just to verify
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
[remote "origin"]
        url = git@git.ee.ethz.ch:alluca/class-fs19.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
        remote = origin
        merge = refs/heads/master
a@debian:~$
```

Tell git your name+email, and tell git the location of the original repository

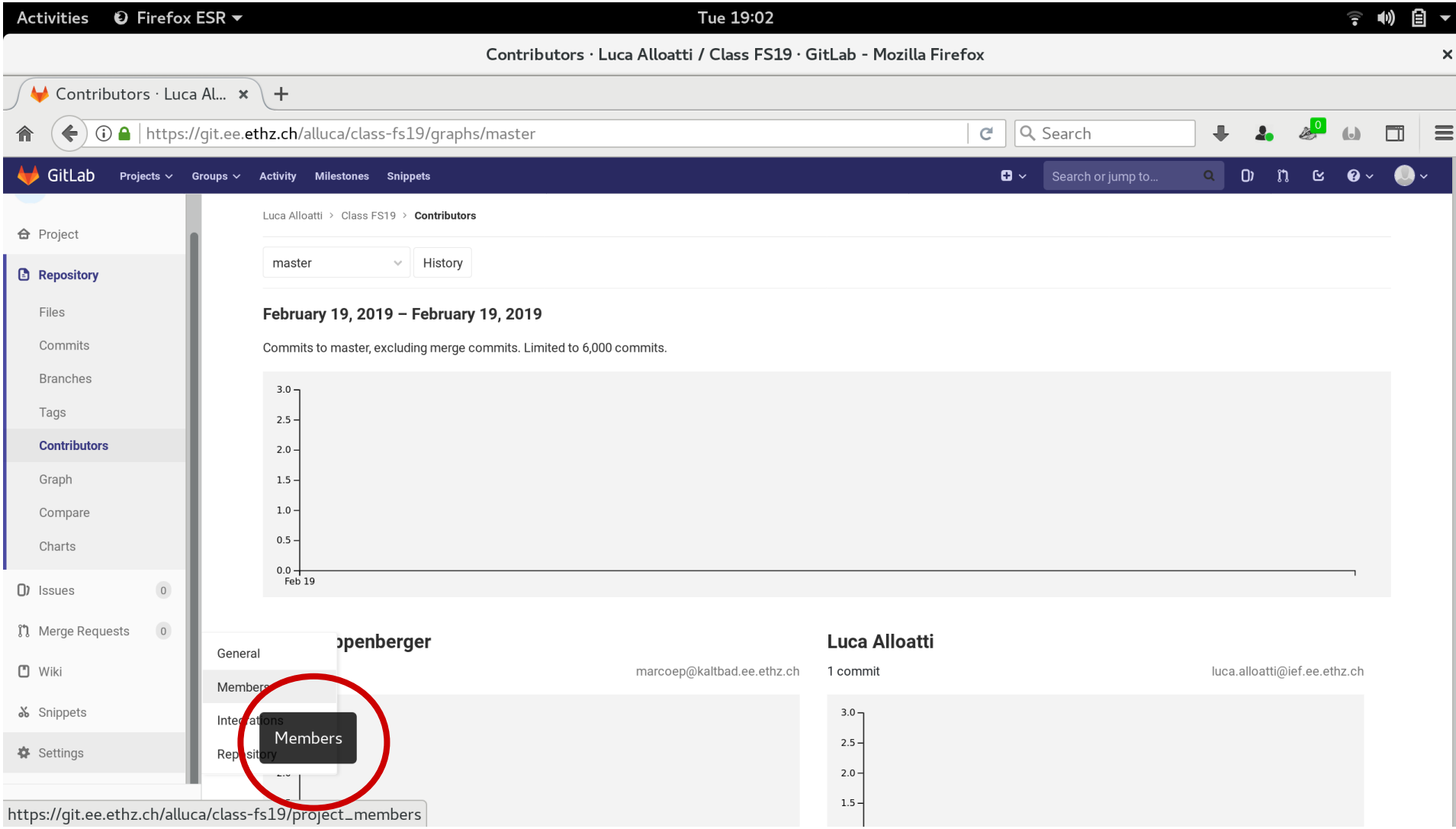
- Follow and adapt these steps to your case:

a@debian: ~/class-fs19

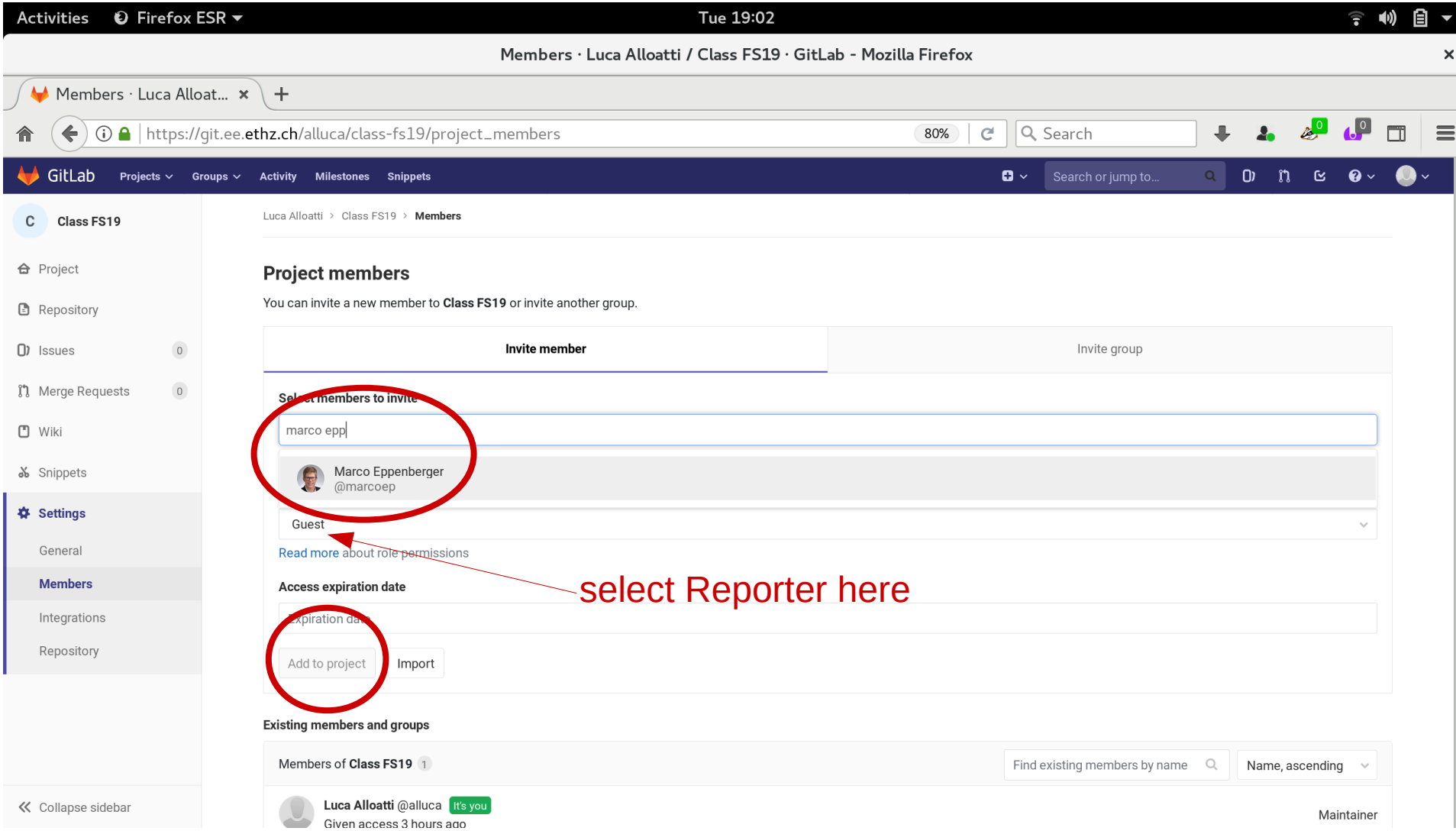
File Edit View Search Terminal Help

```
a@debian:~$ cd class-fs19/
a@debian:~/class-fs19$ git config user.name "Luca Alloatti"
a@debian:~/class-fs19$ git config user.email "luca.alloatti@ie
f.ee.ethz.ch"
a@debian:~/class-fs19$ git remote add upstream git@git.ee.ethz
.ch:python-for-engineers/class-fs19.git
a@debian:~/class-fs19$ cat .git/config # just to verify
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
[remote "origin"]
        url = git@git.ee.ethz.ch:alluca/class-fs19.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
        remote = origin
        merge = refs/heads/master
[user]
        name = Luca Alloatti
        email = luca.alloatti@ief.ee.ethz.ch
[remote "upstream"]
        url = git@git.ee.ethz.ch:python-for-engineers/class-fs
19.git
        fetch = +refs/heads/*:refs/remotes/upstream/*
a@debian:~/class-fs19$
```

Add us to project members



Add us to project members



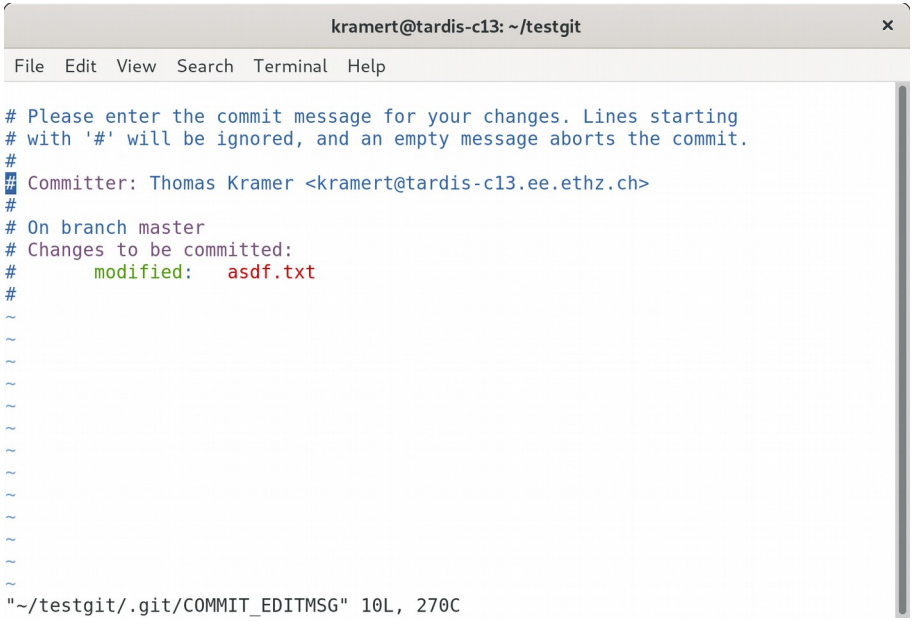
Test that everything works

- Create a test file (for example with the text editor nano)
- Add the new file to the “tracked” list of files:
  - `git add <filename>` # this gives you more control, or use:
  - `git add -A` # this adds ALL files (less control)
- give title to your “*commit*” (-m), then *push* your local copy to the *master* branch of the remote server (the *origin*):

```
a@debian: ~/class-fs19
File  Edit  View  Search  Terminal  Help
a@debian:~/class-fs19$ git add -A
a@debian:~/class-fs19$ git commit -m "My first commit"
[master 9ca0331] My first commit
 1 file changed, 2 insertions(+)
 create mode 100644 MyFistFile
a@debian:~/class-fs19$ git push origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 284 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To git.ee.ethz.ch:alluca/class-fs19.git
   1561fe6..9ca0331  master -> master
```

Frequent trouble maker: git commit without ‘-m’

- Without the argument ‘ -m “My message” ’ git will open a terminal-based text editor to let you enter the commit message.
- The editor is called ‘vim’
  - ‘vim’ is controlled with keyboard shortcuts
    - ‘i’ - Insert text
    - ‘ESC :wq ENTER’ - Leave insert mode, then write to file and quit



Pull any update made to the original repository (from the “upstream”)

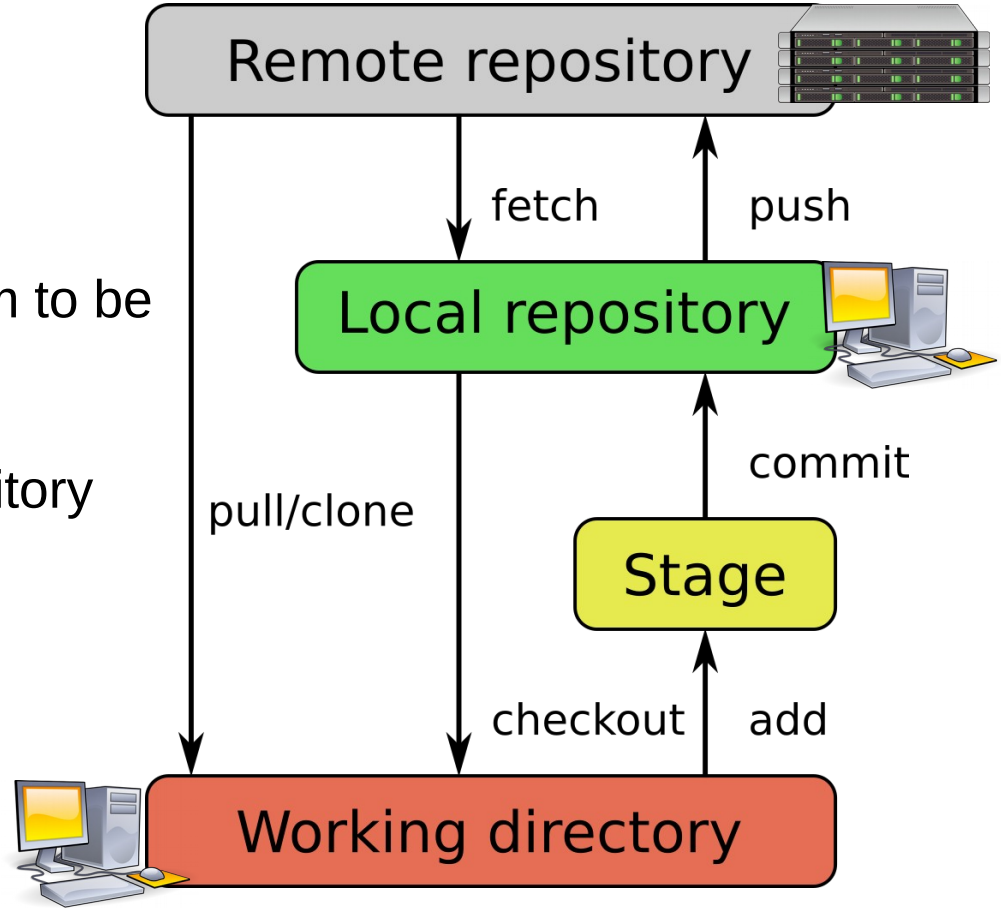
a@debian: ~/class-fs19

File Edit View Search Terminal Help

```
a@debian:~/class-fs19$ git pull upstream master
From git.ee.ethz.ch:python-for-engineers/class-fs19
 * branch            master     -> FETCH_HEAD
Merge made by the 'recursive' strategy.
 Lecture_1/cheatsheet_git.pdf | 130423 ++++++++++++++++++++++
 1 file changed, 130423 insertions(+)
 create mode 100644 Lecture_1/cheatsheet_git.pdf
a@debian:~/class-fs19$
```

Git workflow

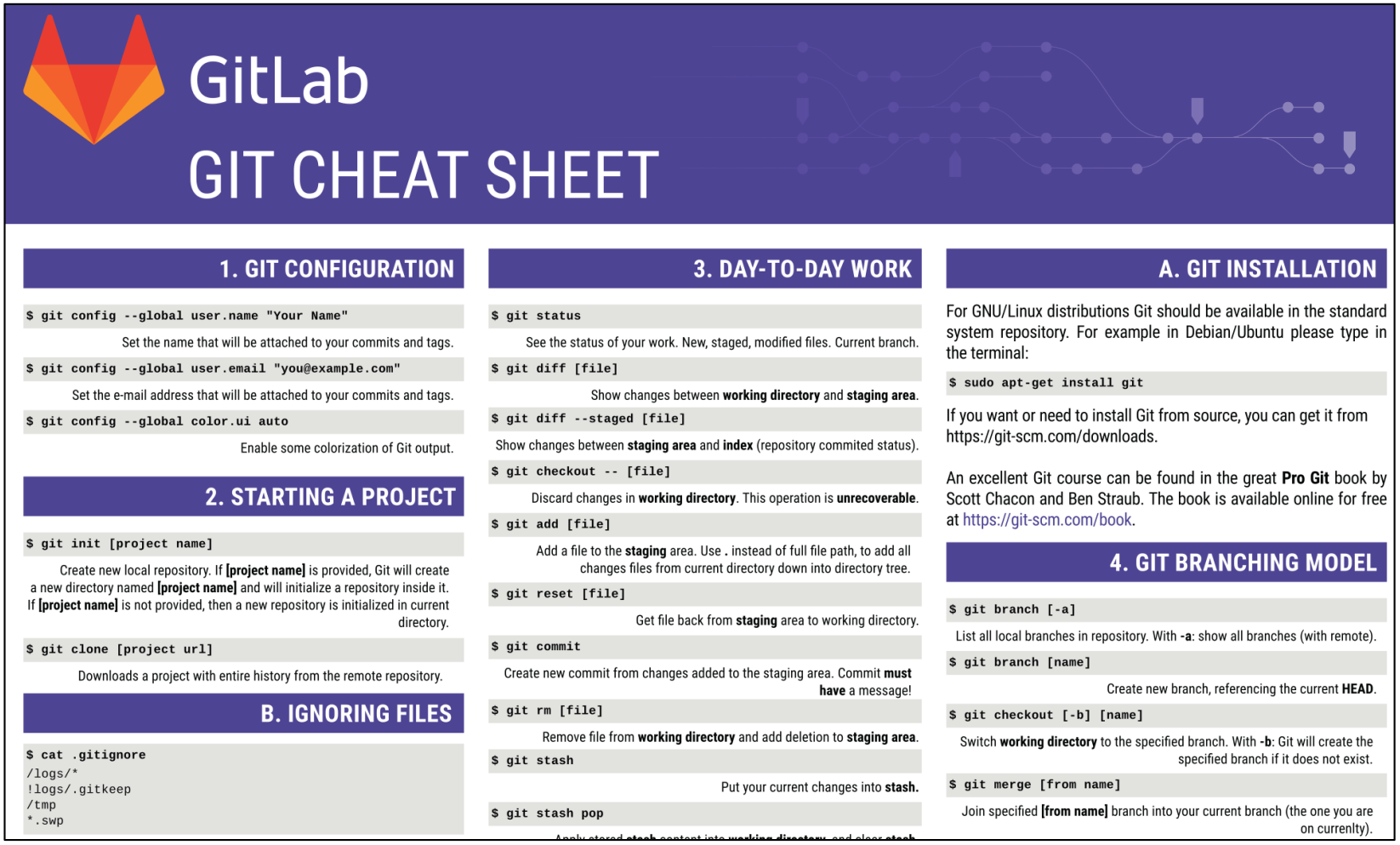
- Obtain a repository
  - *`git init` or `git clone* [url]`
- Make changes
- Stage your changes (mark them to be included in the commit)
  - *`git add myfile.py`*
- Commit changes to local repository (make a ‘snapshot’)
  - *`git commit -m “My message”`*
- Push changes to remote
  - *`git push remotename branchname`*



Copied from: https://marine.rutgers.edu/~cfree/wp-content/uploads/git-and-github-workflows-12-638-300x169.jpg

Familiarizing git commands (in the terminal)

- The file cheatsheet_git.pdf contains the most common git commands
- In the next slides we will practice a few



GitLab

GIT CHEAT SHEET

1. GIT CONFIGURATION

```
$ git config --global user.name "Your Name"
```

Set the name that will be attached to your commits and tags.

```
$ git config --global user.email "you@example.com"
```

Set the e-mail address that will be attached to your commits and tags.

```
$ git config --global color.ui auto
```

Enable some colorization of Git output.

2. STARTING A PROJECT

```
$ git init [project name]
```

Create new local repository. If **[project name]** is provided, Git will create a new directory named **[project name]** and will initialize a repository inside it. If **[project name]** is not provided, then a new repository is initialized in current directory.

```
$ git clone [project url]
```

Downloads a project with entire history from the remote repository.

B. IGNORING FILES

```
$ cat .gitignore
/logs/*
!logs/.gitkeep
/tmp
*.swp
```

3. DAY-TO-DAY WORK

```
$ git status
```

See the status of your work. New, staged, modified files. Current branch.

```
$ git diff [file]
```

Show changes between **working directory** and **staging area**.

```
$ git diff --staged [file]
```

Show changes between **staging area** and **index** (repository commited status).

```
$ git checkout -- [file]
```

Discard changes in **working directory**. This operation is **unrecoverable**.

```
$ git add [file]
```

Add a file to the **staging** area. Use **.** instead of full file path, to add all changes files from current directory down into directory tree.

```
$ git reset [file]
```

Get file back from **staging** area to working directory.

```
$ git commit
```

Create new commit from changes added to the staging area. Commit **must have** a message!

```
$ git rm [file]
```

Remove file from **working directory** and add deletion to **staging area**.

```
$ git stash
```

Put your current changes into **stash.**

```
$ git stash pop
```

Apply stored **stash** content into **working directory**, and clear **stash**

A. GIT INSTALLATION

For GNU/Linux distributions Git should be available in the standard system repository. For example in Debian/Ubuntu please type in the terminal:

```
$ sudo apt-get install git
```

If you want or need to install Git from source, you can get it from https://git-scm.com/downloads.

An excellent Git course can be found in the great **Pro Git** book by Scott Chacon and Ben Straub. The book is available online for free at https://git-scm.com/book.

4. GIT BRANCHING MODEL

```
$ git branch [-a]
```

List all local branches in repository. With **-a**: show all branches (with remote).

```
$ git branch [name]
```

Create new branch, referencing the current **HEAD**.

```
$ git checkout [-b] [name]
```

Switch **working directory** to the specified branch. With **-b**: Git will create the specified branch if it does not exist.

```
$ git merge [from name]
```

Join specified **[from name]** branch into your current branch (the one you are on currenlty).

Create a new branch and push it

Micro-Exercise (5 min)

- `cd <where you cloned the repository>/class-fs20`
- `git checkout -b MyTestBranch` (this switches to a new branch)
- `git branch` (check to be in the “MyTestBranch” branch)
- create a new file, for example “BranchTest.txt”
- `git add BranchTest.txt` (this tells git to track this particular file)
- `git commit -m “My branch-test commit”`
- `git push origin MyTestBranch` (push new branch to server)
- navigate to the https://git.ee.ethz.ch and verify that your commit is present (on the left menu: _Repository _Graph)
- `git checkout master` (switch back to the master branch)
- `ls` (verify that BranchTest.txt is not there)
- `git pull origin MyTestBranch` (this merges the test branch into the master)
- `ls` (verify that BranchTest.txt is present)
- push the new changes to master and check the graph again on the web interface, or type “`git log --graph`” or “`gitg`”

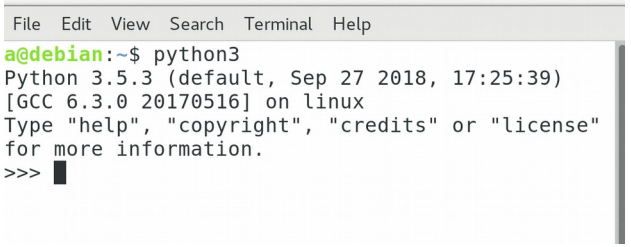
End of git setup and exercises
-please wait for your colleagues to finish-

My first python program: print(“hello world“)

Four ways of running python scripts

Micro-Exercise (5 min)

- In the following we will run “print(“hello world”) using four different programs:
  - the **jupyter-notebook**: it runs in a browser and will be used most of the time during the lecture. It allows to visualize the outputs in the same browser and to run independent “cells” of code similarly to Mathematica
  - from the **command-line** by typing “python3 FileName.py”
  - Python **interactive shell**: after typing “python3” in a terminal the “python interactive shell will open where commands can be typed and executed directly
  - An **IDE**: Typically, full IDEs support more functions than jupyter and it allow interactive debugging. Examples: **pycharm** or Eric6.



First necessary step before using python in this course: Setup the environment

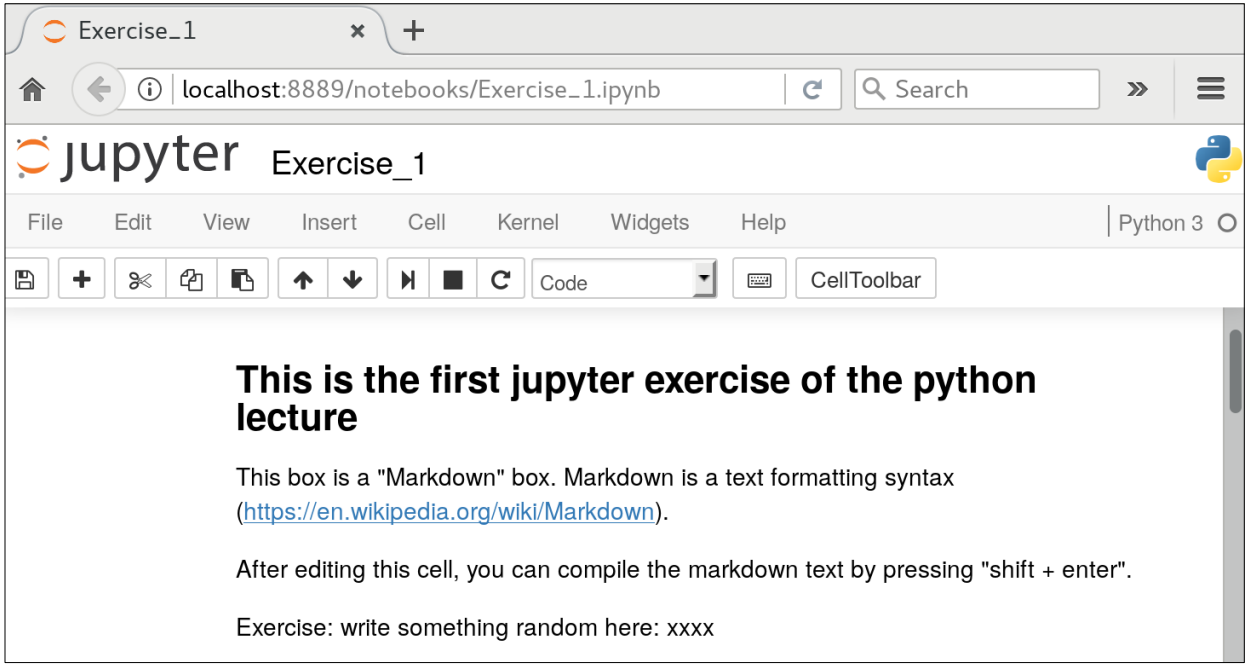
- in a terminal (press “window” key then type “terminal”…) type the following:

```
source /usr/pack/anaconda-3-fg/anaconda3_env.sh
```

- the command above will make available some special programs (jupyter-notebook, python 3.7 and pycharm IDE) which are not installed by default even without being root users
- If successful, `(base)` will appear in front of every line in your console
- this command must be repeated at **every new lecture** and in **every new terminal** (when needing python)

Method 1: jupyter-notebook

- In a terminal, type “jupyter-notebook &” (the “&” will keep the terminal active)
- Within jupyter, open the file “Exercise_1” and follow the instructions in it

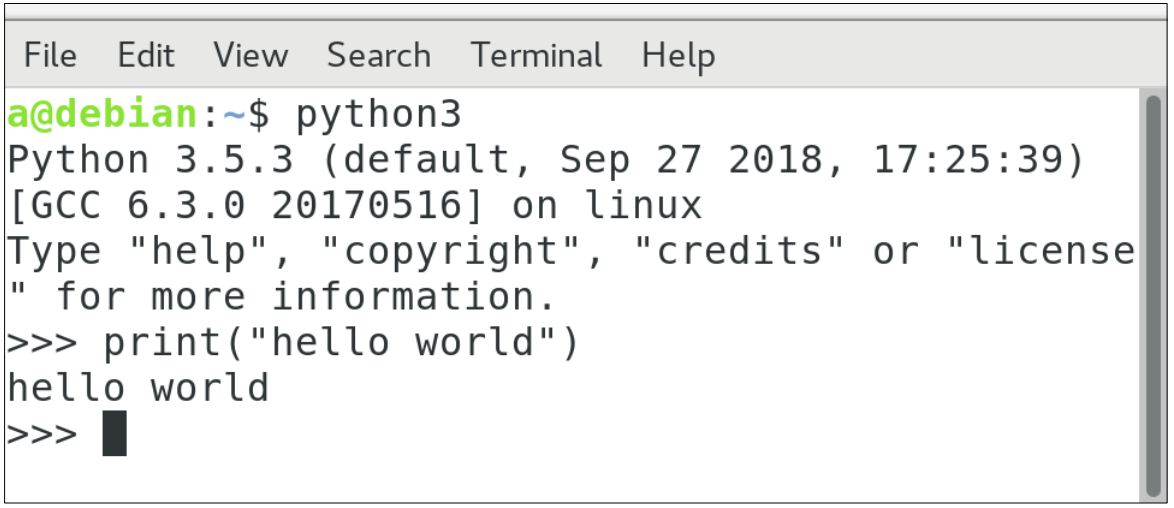


Method 2: run python on a .py file (command line)

- create a file “hello.py” containing the following text:
  `print(“hello world”)`
- run it by typing in the terminal:
  `python3 hello.py`

Method 3: python interactive shell

- In a terminal, type “python3”. This will launch the python interactive shell
- In the shell, write `print(“hello world”)` and press enter
- To quit the shell: “Ctrl + d”

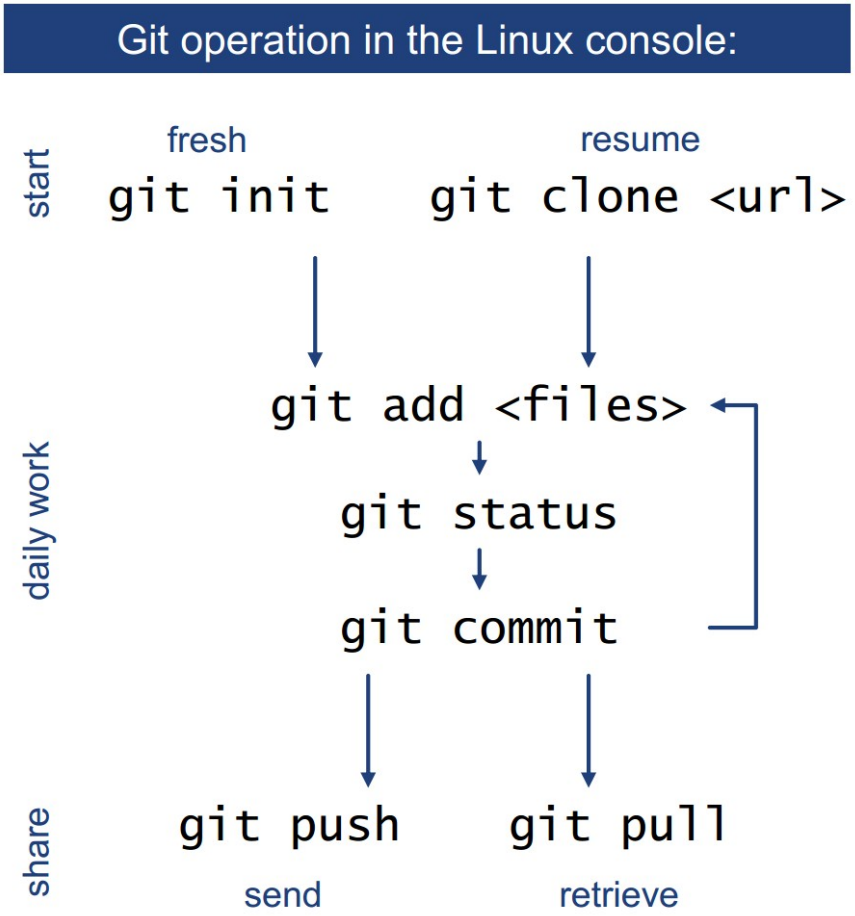


Summary: command-line commands

- Refer to the command manual (`man <command-name>`) whenever in doubt, or to the cheatsheet_linux_command_line.pdf
- frequent commands: `ls, pwd, mkdir, rm, top, grep, kill, ssh, cp, mv`
- Tab: autocomplete, Ctrl+c: stop current command, window+arrow: arrange windows, window+tab: switch windows, window+`: switch windows of a same program, Ctrl+Enter: launch a second instance of a program

Summary: git

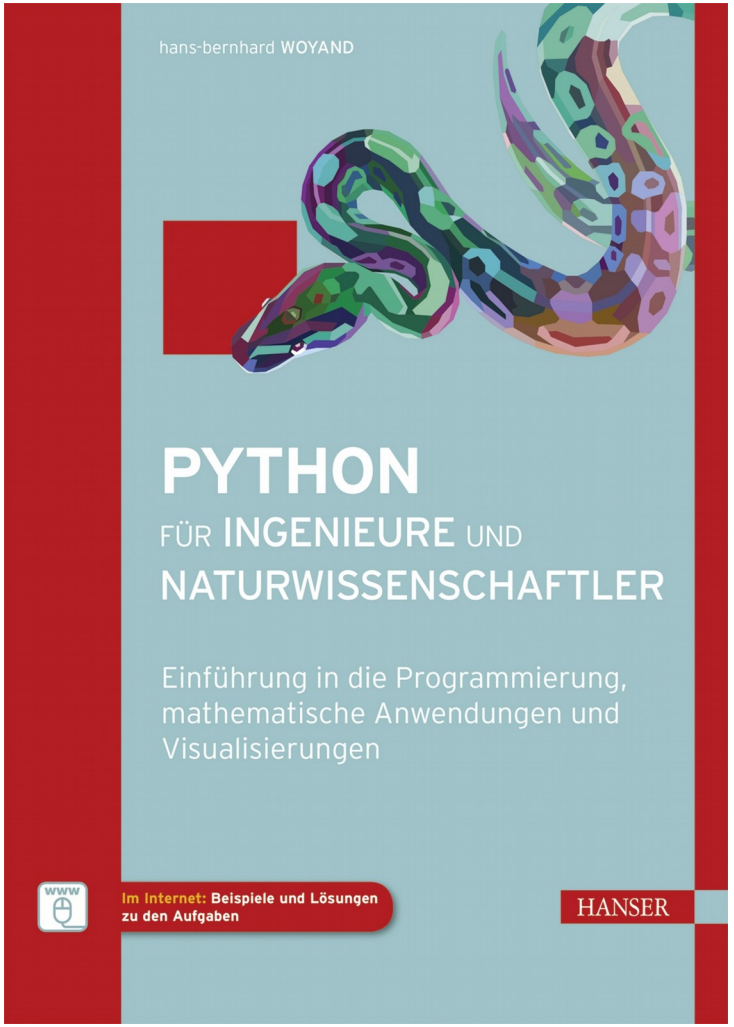
- Git is a hierarchical, decentralized source-code management system.
- Keep track of changes of human-readable files
  - source code, scripts, TeX
- Keep old versions of each file with possibility to restore.
- Collaborate with others on a large scale
  - Git was devised by Linus Torvalds to coordinate work on the Linux kernel with thousands of contributors
- Fine-grained control over conflict resolutions



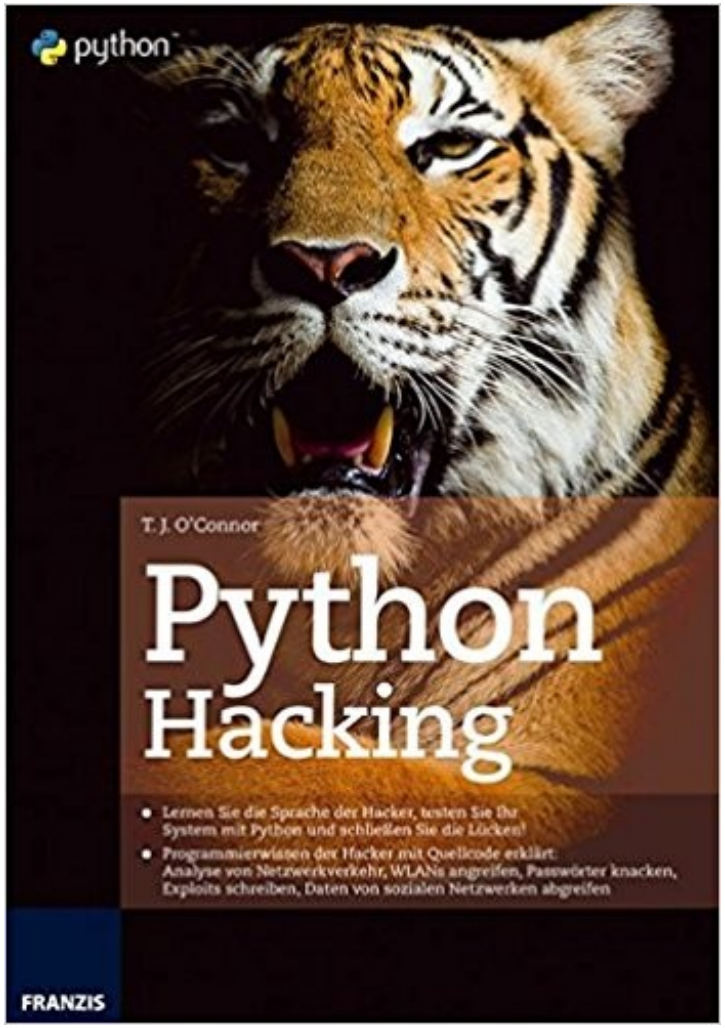
Extra literature

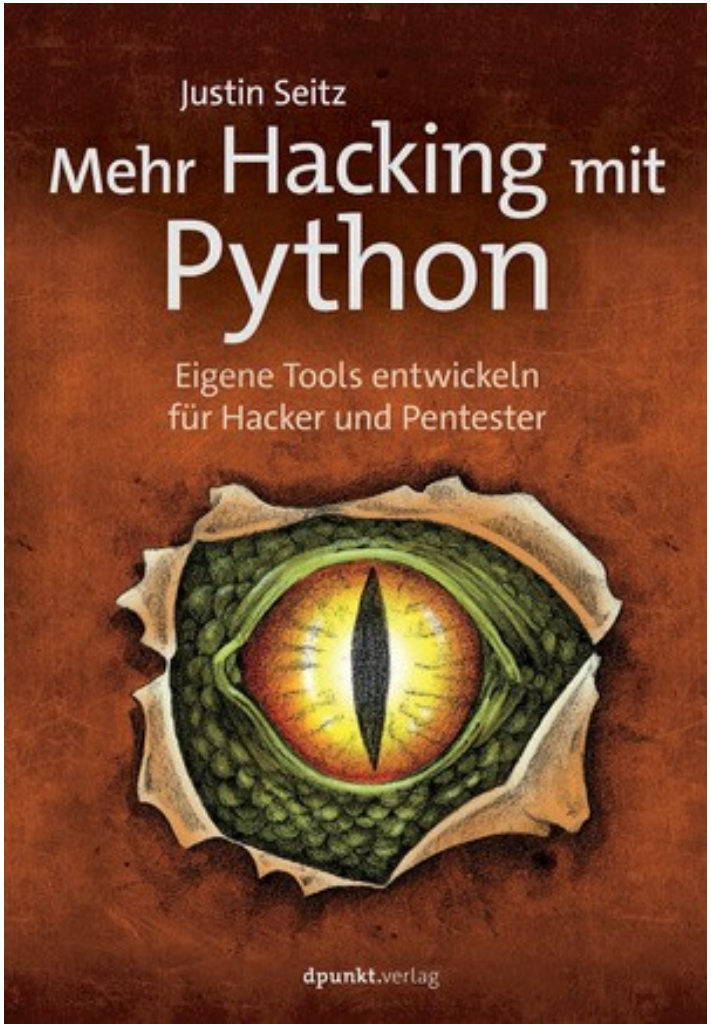
There are several books both on python and on Linux. In the following is given a small (subjective) selection

Books: introduction to python

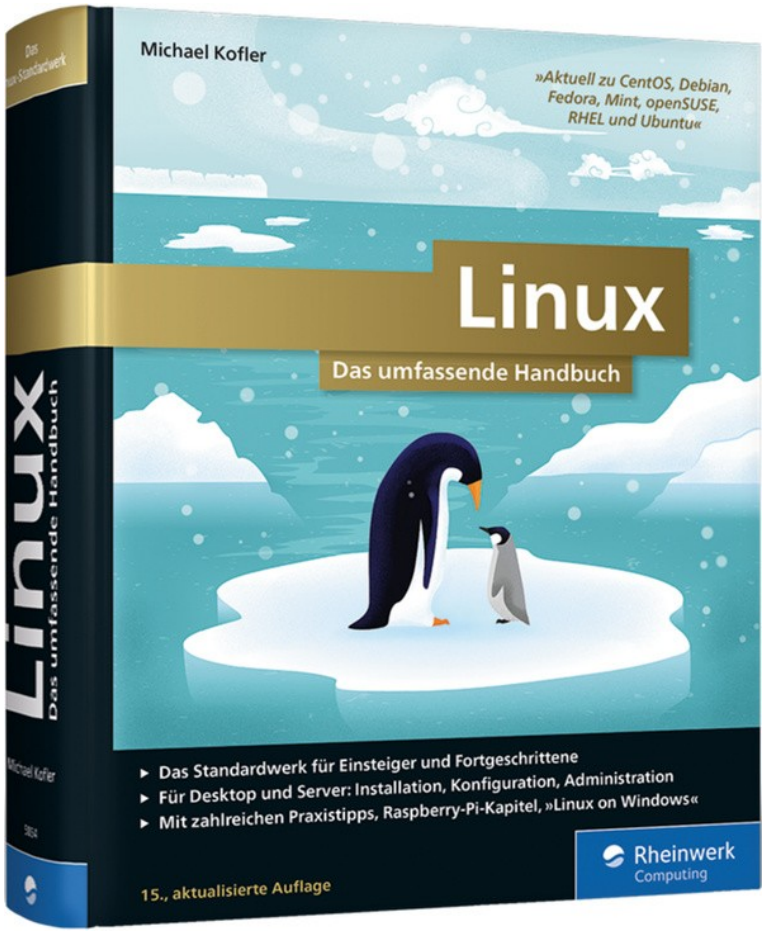


More books on python...





Book on Linux and Unix



Introduction to Linux

A Hands on Guide

Machtelt Garrels

http://www.tldp.org/LDP/intro-linux/intro-linux.pdf

Unix Power Tools, 3rd Edition

By M. Loukides, T. O'Reilly, J. Peek, S. Powers

O'Reilly Media

Finally, before leaving the classroom:

- commit and push your last changes

Micro-Exercise (1 min)

a@debian: ~/class-fs19

File Edit View Search Terminal Help

```
a@debian:~/class-fs19$ git add -A
a@debian:~/class-fs19$ git commit -m "My first commit"
[master 9ca0331] My first commit
 1 file changed, 2 insertions(+)
 create mode 100644 MyFistFile
a@debian:~/class-fs19$ git push origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 284 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To git.ee.ethz.ch:alluca/class-fs19.git
   1561fe6..9ca0331  master -> master
```