Lab #1: Introduction, agenda, GUI and GitLab

Introduction to Linux (NSWI177)

NSWI177 Home
Contact
Goals
Grading
USB Disk
Resources
Dual-boot
Labs & Lectures
Mini manual
Q & A

Labs: **1**, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14.

Table of contents

The primary goal of this lab is to ensure all of you have a working GNU/Linux installation available and that you can sign in to MFF GitLab. You will also select a desktop environment and recap some software engineering principles and best practices.

However, first we will describe course completion requirements, semester organization, etc.

Labs organization

Because the entry knowledge of students for this course varies a lot, we decided to organize the labs in a self-study manner and use **Zoom** for consultation.

Invitations to the Zoom rooms will be available via SIS and in your GitLab repository (more about these later).

For each week (lab), we will publish examples, exercises, and other materials on this website. The exercises will be always accompanied by our solution so that you can check and compare it with yours. We expect you will come to the lab if you encounter an issue that you are unable to solve by yourself. In other words, the on-line lab will be a moderated session of questions and answers.

We believe this should prove to be the most time-efficient organization for both you and us. If you already know the topic, you can blaze through the exercises in a few minutes and be done with it. However, if you run into trouble, we are here to help. Do not hesitate to ask.

We will stress this once again – do not be afraid to ask if you run into trouble. Describe what you tried to do, what worked and what did not and talk to us.

Please note that all labs (on-line consultations) are the same – if you cannot attend the one you are enrolled on, you can visit any other one. Please, do this in extraordinary cases only to keep the number of enrolled students balanced across all labs. Thank you!

Grading and deadlines

Grading is described in more detail on a **separate page**.

Briefly, there will be graded tasks for each lab, from each lab you need to get at least 20 points out of 100. Overall grade will be computed from the total sum of all points across all labs.

Solutions must be submitted in time, we will not accept any late submissions. We will be using GitLab as the submission system.

There are graded tasks even for this lab – please, see the end of this page.

Your GNU/Linux installation

Under *normal* circumstances, you would have access to machines with GNU/Linux installations at school. With remote teaching, you need to install Linux yourself on your own computer.

Because the installation may seem daunting (although it is definitely not), we prepared for you USB disks with a preinstalled system.

Please follow the instructions on **USB disk page** to obtain and prepare your own disk.

Return to this page once you boot from the USB (or using any other method mentioned) to continue with this lab. We strongly recommend you try to prepare the disk before the lab and use the lab if you encounter any issues. Coming to the lab without having the image at least downloaded is a waste of everybody's time.

Before booting into GNU/Linux

If possible, we recommend you open this page on your cell phone and use it to guide you through your first boot before you will be able to display this page in your Linux.

Selecting your desktop environment

Once you boot from the USB disk, you can choose which desktop environment you will use.

On most operating systems, there are not many options on how to control your graphical interface. With GNU/Linux, there is a much wider choice and you choose between rich environments with plenty of eye-candys to very austere ones that do not even employ a mouse (and dozens of environments somewhere in between).

On your installation, you can choose from six different environments.

GNOME, **Plasma** (a.k.a. KDE), **Xfce** and **LXDE** represent mainstream desktop environments that should be familiar for use for any user coming from either Windows or Mac environments. The above ordering roughly corresponds to the amount of eyecandy they offer (and to the hardware requirements they need).

Openbox and **i3** are special environments as they do not contain the traditional task bar with a list of windows and require a bit more patience before they are mastered. On the other hand, the time investment, especially for i3 that is driven by keyboard only, pays back in a much more efficient usage of your computer.

We encourage you to try all of them. Login into the environment, determine how applications are launched and decide which environment you like the most. Note that the environments can be further customized – from the overall color scheme to keyboard shortcuts.

If you are unable to decide, Plasma is a good choice for ex-Windows users with decent hardware. Choose LXDE if your machine is shorter winded. And after a month of using these, switch to i3 to become a true power user.

Selecting your applications

Once you decide on your desktop environment, look around for other applications you will need.

Above all, look around for the text editors available. There are several popular graphical editors *already* installed as you can see on the following screenshot.

Note that other editors are available from the command-line: we will talk about these during the next lab.

GitLab …

We will now step away from Linux itself and devote our attention to GitLab. However, keep reading this inside your Linux environment :-).

We will start with a bit of motivation why you want to learn about GitLab and then we will dive into it.

Modern software is rarely a product of one–man teams. Rather, it is developed by large teams that can span several time zones or even continents.

Development in such teams requires that all developers have access to the (most up-to-date version of the) source code and that they can communicate with other members of the team efficiently.

There are many solutions to this: from e-mails and shared network disks to more sophisticated solutions. To prepare you a little for the software engineering practice, we will be using one of the more sophisticated solutions and that is **GitLab**.

GitLab offers a place where developers can share the source code but also manage a list of existing bugs, keep documentation, and even automatically test their code. And since it can be integrated with other tools as well, for many companies as well as open source projects, GitLab became the central place for their product.

And we can use its advantages even when working alone. Even if we would use it only as a smart backup for our source code.

… and Git

GitLab itself is built around **Git**. Git is a versioning system. In layman terms, it means that it watches your files for changes and remembers previous versions of your files. It has the big advantage that you can freely update your code and still return to its older versions.

We will dedicate one of the following labs to Git fully. Take this description as a very high-level overview so you can start working with Git the GUI-way in GitLab.

Git does not track all files as there is typically no need to version the compiled files (because you can always recreate them). For example, for Java project you do not need to track `*.class` files as you can create them from `*.java` ones by compiling the source codes again (some applies to `*.pyc` with Python or `*.o` with C++). Another example would be that you do not track PDF export of a LibreOffice document (though tracking `*.odt` files is not something where Git would unleash its full potential).

Git does not create the versions *automatically* as each version is supposed to capture a reasonable state of the project. Thus, for example, you create a new version (sometimes also called revision) once you add a new feature to your software. Or when you fix a bug. Or when you fix a typo in the documentation. It allows you to create a reasonable history of the software that is small enough for reviewing (for example) but does not capture every small typo you made (it does not serve as a *undo/redo* replacement of your editor).

And when employed in a team, Git allows the cooperation of multiple users as it can merge unconflicting changes without manual intervention. For example, if Alice makes a change to file `alpha.txt` and Bob at the same time changes the file `bravo.txt`, Git allows Carol to pull changes from both of them at the same time without any fear of overwriting the changes done by either of them. Again, you will try this yourself in one of the next labs, at the moment you will be using only the web-based GUI. And since you will be the only user doing changes, there is no need to worry about possible conflicts.

With MFF GitLab

For this course, we will be using the faculty instance of GitLab at **https://gitlab.mff.cuni.cz**. Please, do not confuse it with the instance at **https://gitlab.com** that you can freely use but is in no way connected with this course.

For login (username) you will be using your **CAS credentials**, i.e., the same ones as you use for SIS. Your first login will activate your account.

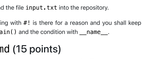
Please **activate your account now**, if you have not yet done so.

First steps inside GitLab

To quickly try GitLab (we will focus on it more in several labs), create a new project (create a *Blank project*). You need to fill in a project name, its slug that is used in the URL, and its visibility. In the example screenshots below, we create a project with our source code from the introductory programming course. Do not forget to ensure that the project is initialized with *README*.

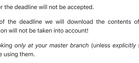
Now open *Web IDE* which is a simple editor available for on-line editing of the source code files.

Using the icons and the help of the following set of screenshots, create a new file, name it `hello.py` and insert a simple Python program.



We will now create a so-called **commit**. Commit in Git captures the current state of the project and can be seen as a named version. In fact, whenever you create a commit, Git will ask you for a **Commit message** where you are supposed to describe what changes you made. We highly recommend the article **How to Write a Git Commit Message** by Chris Beams for nice tips on how to write a good commit message. However, it might make more sense to return to this article later on once you know Git a bit more.

For now, we will be making all changes directly to the *Master branch*. We will explain the concepts of branches later on, for now take them as a magic that works :-).



Sign-out from GitLab now.

On your own: sign in to GitLab again, find your project and create a new file, pasting in some of your source code. Note that when you click on the filename on the project homepage, you will see its contents and again a link for its editing.

Graded tasks

The graded tasks are already prepared in your GitLab repository for this course. Please refer to the **grading page** for details.

Activate your GitLab account

You have already done that, right? Then you can continue with submission of the actual tasks.

Update The repositories for your submission will be created at the beginning of March (we are still debugging some details).

Update #2 We can assign you to the project *only after* you have activated your account. Hence, there will be some delay between your account activation and before you can actually access this repository.

`01/gif.md` (30 points)

This is a quiz that should check your basic understanding of file contents and storage of binary values. Please, answer directly into the file.

`01/longest.py` (20 points)

Modify the program so that it prints the longest line (use `len` as we do not care at the moment about ligatures, code points, etc.).

Important: even if you decide you will not use the skeleton from us, do not upload the file but always modify it. Otherwise, you will overwrite its executable attribute (more about that later) and it will break the tests.

Note: the first line starting with `#!` is there for a reason and you shall keep it there (more details on some of the next labs). Same for the usage of `main()` and the condition with `__name__`.

Update: the loop that is already present in the source file iterates over all lines of input. (i.e. you do not need to add call to `input()`). Once you launch this program, you can enter individual lines. To terminate the input, hit `Ctrl-D` (or `Ctrl-Z` if you need to run it in Windows).

`01/prime.py` (20 points)

Again, modify the program to print whether a number in the file `input.txt` is a prime number.

Important: even if you decide you will not use the skeleton from us, do not upload the file but always modify it. Otherwise, you will overwrite its executable attribute (more about that later) and it will break the tests.

Important: do not upload the file `input.txt` into the repository.

Note: the first line starting with `#!` is there for a reason and you shall keep it there (more details on some of the next labs). Same for the usage of `main()` and the condition with `__name__`.

`01/survey.md` (15 points)

Not a test, we are only interested in what hardware and software you are using :-).

`01/env.txt` (15 points)

Which desktop environment have you chosen and why?

Please, answer in full sentences that make sense and are without typos. We expect about 1-2 reasonable paragraphs of text. Feel free to write on how you have customized the environment.

We are accepting answers in English, Czech or Slovak language.

Deadline: March 22, AoE

Solutions submitted after the deadline will not be accepted.

Note that at the time of the deadline we will download the contents of your project and start the evaluation. Anything uploaded/modified later on will not be taken into account!

Note that we will be looking *only at your master branch* (unless *explicitly specified* otherwise), do not forget to merge from other branches if you are using them.

Department of Distributed and Dependable Systems
Faculty of Mathematics and Physics
Charles University
Malostranské náměstí 25
118 00 Praha 1
Czech Republic