Introduction to `git`

Slides partially taken by Pasquale Africa, Konstantin Karchev and Dario Coscia

Notes 2024 part1 Notes 2024 part2

Notes 2025

Installation

https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

- **Linux** Typically in Linux systems is already installed, try `git --version`. If it's not installed you can do `sudo apt install git-all`
- **macOS** Try `git --version` if not installed it will prompt you to install it.
- **Windows** find you executable on https://git-scm.com/downloads/win

Command Line vs GUI

There are several way to use `git`.

- **Command Line** one can use it directly from the command line and all commands are available there. If you are not familiar with the command line it will take some time, but it has many advantadges. Look for Terminal in macOS and Linux, and Command Prompt or PowerShell in Windows.
- **GUI version** Graphical user interface versions are available everywhere online, you can choose one of those. They are not always complete, but they allow to a have a simpler and graphical interface.

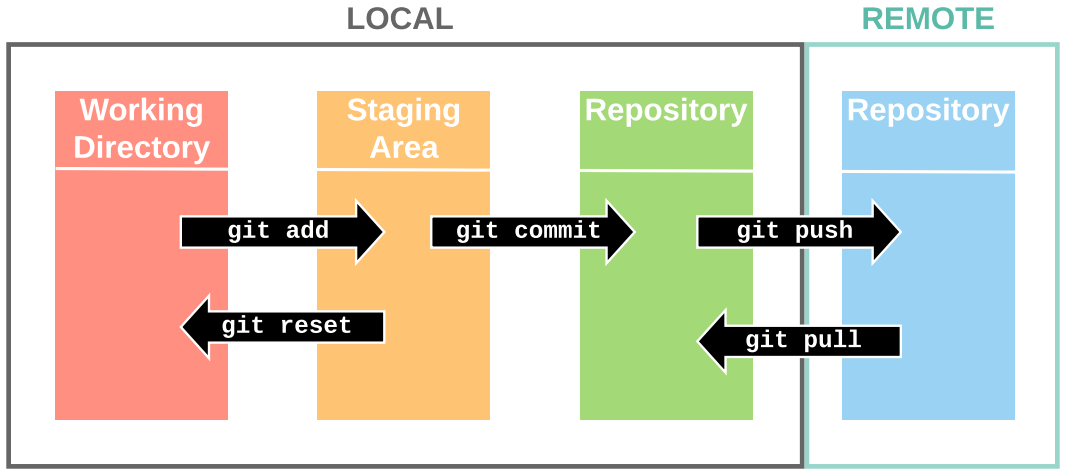
Version control

Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time and prevent/mitigate disasters (code review, reverting, and patching).

`git` is a free and open-source version control system, originally created by Linus Torvalds in 2005. Unlike older centralized version control systems such as SVN and CVS, Git is distributed: every developer has the full history of their code repository locally. This makes the initial clone of the repository slower, but subsequent operations dramatically faster.

How does `git` work?

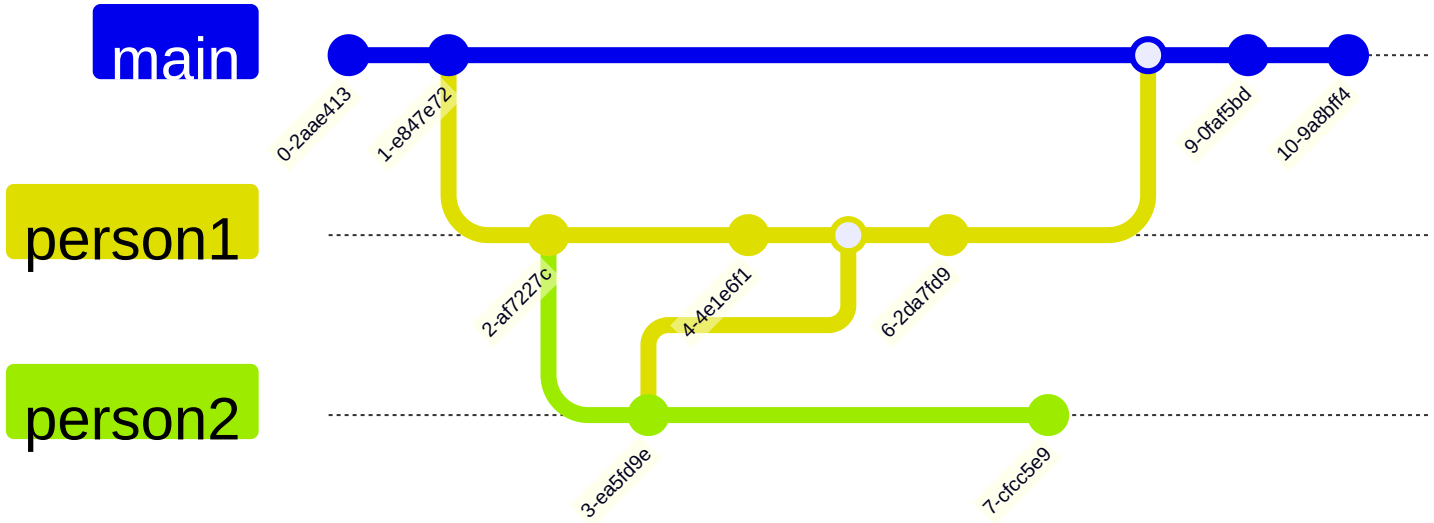
1. Create (or find) a repository with a git hosting tool (an online platform that hosts you project, like GitHub or Gitlab).
2. `git clone` (download) the repository.
3. `git add` a file to your local repo.
4. `git commit` (save) the changes, this is a local action, the remote repository (the one in the cloud) is still unchanged.
5. `git push` your changes, this action synchronizes your version with the one in the hosting platform.



How does `git` works? (Collaborative)

If you and your teammates work on different files the workflow is the same as before, you just have to remember to `pull` the changes that your colleagues made.

If you have to work on the same files, the best practice is to create a new `branch`, which is a particular version of the code that branches form the main one. After you have finished working on your feature you `merge` the branch into the main.



Other useful `git` commands

- `git diff` shows the differences between your code and the last commit.
- `git status` lists the status of all the files (e.g. which files have been changed, which are new, which are deleted and which have been added).
- `git log` shows the history of commits.
- `git checkout` switches to a specific commit or branch and switch a specific file to the latest commit version.
- `git stash` temporarily hides all the modified tracked files.

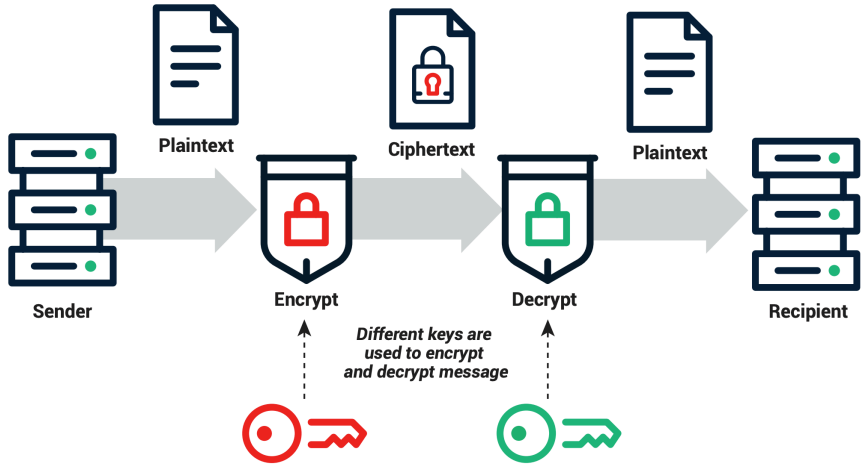
An excellent visual cheatsheet can be found here.

SSH authentication

1. Sign up for a GitHub account.
2. Create a SSH key.
3. Add it to your account.
4. Configure your machine:

```
git config --global user.name "Name Surname"
git config --global user.email "name.surname@email.com"
```

See here for more details on SSH authentication.



Initting

Take a repository project that already exists online, you can `clone` the project.

Example: clone the course repository

```
git clone git@github.com:accdavlo/calcolo-scientifico.git
```

Pulling

Getting update from the repository is simple. If changes have been applied to the project, you can simply do in the working directory

```
git pull
```

Example: before every lecture, you can download the latest updates by running `git pull` from inside the cloned folder.

Exercises

Exercise: hands on `git`. Collaborative file management (1/3)

1. Form groups of 2-3 members.
2. Designate one member to create a new repository (visit https://github.com/ and click the `+` button in the top right corner), and ensure everyone clones it.
3. In a sequential manner, each group member should create a file with a distinct name and push it to the online repository while the remaining members pull the changes.

```
git pull
git add file_name
git commit -m "Message describing what I did in this changes"
git push
```

4. Repeat step 3, but this time, each participant should modify a different file than the ones modified by the other members of the group.

Exercise: hands on `git`. Collaborative file management (2/3)

Now, let's work on the same file, `main.py`. Each person should create a hello world `main.py` that includes a personalized greeting with your name. To prevent conflicts, follow these steps:

1. Create a unique branch using the command: `git checkout -b [new_branch]`.
2. Develop your code and push your branch to the online repository (read what `git` says).

```
git add files
git commit -m "message"
git push
```

3. Use `git branch` to check which branches are available and in which one you are.
4. Once everyone has finished their work, merge your branch into the `main` branch using the following commands:

```
git checkout main
git pull origin main
git merge [new_branch]
git push origin main
```

Exercise: hands on `git`. Collaborative file management (3/3)

How to deal with `git` conflicts

The first person to complete this process will experience no issues. However, subsequent participants may encounter merge conflicts. Read what git proposes! (`git pull`)

Git will mark the conflicting sections in the file. You'll see these sections surrounded by `<<<<<<<`, `=======`, and `>>>>>>>` markers.

Carefully review the conflicting sections and decide which changes to keep. Remove the conflict markers (`<<<<<<<`, `=======`, `>>>>>>>`) and make the necessary adjustments to the code to integrate both sets of changes correctly.

After resolving the conflict, `commit` your changes and push your resolution to the repository.

Other commands / keywords

- `git rebase`
- `git revert`
- `git remote`
- Pull requests
- Forks
- Public libraries
- Testing

Git Game

> NOTE!
> Are you a Gen Z-er who wants to learn Git *and* have heaps of harmless fun at the same time? Check out Oh My Git!