Git

Git is a Version control system.

Initialize Git repository

```
git init
```

to initialize an empty Git repository. It creates an .git folder. **Never** modify this folder.

```
tree .git/
```

to see the content of the folder.

```
git status
```

to see if every file is updated.

```
git add earth.txt
```

to add a file into the **staging area**. The status of the file pass from **untracked** to **staged**.

Config Git

```
git config --global user.name "Matteo Nurisso"
git config --global user.mail "mnurisso@sissa.it"
```

```
git config --global core.editor "vim"
```

to setup Vim as standard editor for commit messages.

Commit

A commit is a screen shot of your working tree at a specif time. Another way to see them is as an increment of the previous work, everyone pointing to the previous one.

```
git commit
```

Open a text editor to edit the commit message. You need to add a commit message in order to do a proper commit. With the commit the file is now in the **repository** .git. Git doesn't see any other file in the working tree, because the file is now considered as unmodified wrt the last repository version.

By modifying a file wrt the repository version the command

```
git status
```

should show it as modified.

```
git diff
```

shows the differences between repository version and local modified version. The file has to be added to the **staging area** by

```
git add filename.txt
```

To commit in a single line:

```
git commit -m "git comment"
```

Commit log list

```
git log
```

shows a list of every commit with the commit comment and a unique identifier.

```
git checkout 946bdd #some characters from the identifier are enough
```

to go back to the specific commit. Move the **HEAD**, a pointer to the commit in which you are.

To show were you are in the commits (in a goodloking way):

```
git log --oneline --graph --all --decorate

* 06d935a (master) add humans
* 946bdd8 (HEAD) add earth.txt
```

To go back to the newest version (of a specific branch):

```
git checkout master #master is the branch name
```

Recover file

```
git checkout -- .
```

to recover files if you by mistake delete important files. If you want to recover only a single file:

```
git checkout -- filename.txt
```

Change filename

If a file is renamed:

```
git add oldname.txt newname.txt
```

In this way git is able to compare the content of the deleted and new file and to recognize that a filename is changed.

```
git mv oldname.txt newname.txt
```

Automatically add the change to the **staging** area.

Github

You can connect your repository trough **ssh** or **https**. With the first one you don't need to put username and password every time, it uses the public key to generate and encripted message that can be decripted only with your private key.

Clone

```
git clone "repository adress"
```

Create a folder with the same name of the repository on your local directory.

```
git remote -v

origin  https://github.com/asartori86/advanced_programming_2019-20.git (fetch)
origin  https://github.com/asartori86/advanced_programming_2019-20.git (push)
```

The default name for the address is **origin**, so that it can be used in commits as shortcut for the standard online address.

Pull

The actual pull command performs two different commands.

```
git fetch origin
```

Fetch downloads the changes but they're stored only in the **.git repository** but without modifying your local files.

```
git merge origin/master
```

To actually apply modifications to your local directory.

```
git pull origin master
```

does both the commands at the same time.

Fork

Fork a repository means that you're creating a repository under your username with the material of the original one. It takes a snapshot of the original repository but then is up to you to keep the repository updated with the original one.

You can add a remote link to the local folder by:

```
git remote add myfork https://github.com/mnurisso/advanced_programming_2019-
20.git
```

So in this way I can have more than one remote link to the local folder, one without writing permission to clone the updates from the original forked code and one with permission to upload and keep track of modifications that I do to the original code.

Push

```
git push myfork master #myfork is usually origin with only one remote
```

By using **http** it will require username and password.

Branch

Branch is a pointer to a commit. The branch always point to the last commit done on the specific branch.

To create a new branch:

```
git branch moon
```

To see the list of branches:

```
git branch

* master
  moon
```

After the creation of a branch I'm still not in the new branch, in order to do that I need to use the **checkout** command.

```
git branch newbranch startingbranch #e.g. jupiter master
```

The branch is created from the actual position or adding another branchname to the branch command.

A general rule for branches is: new feature = new branch.

Merge

```
git checkout master #final branch
git merge branch_name
```

Fast-forward if there is nothing else apart from pushing forward the master branch. If it's not fast-forward a commit message is necessary to explain why the merge is necessary.

```
git branch -d branch_name
```

Delete a branch, but it's only a pointer so it's not deleting the commitments.

More info

```
man git-commandname
```

to have all the infos about a specific git command.