git

What it is and why you should really use it

What is version control, anyway

A tool that keeps track of the changes to the code and can:

- Restore an older version, show what was changed, when and by whom
- Keep a copy on a server that survive when your computer is gone
- Allow many people to work on the same project
- Allow to explore variations of the codebase in parallel and later abandon/merge them

Why git

It is basically the standard.

There are others: SVN, Mercurial, Fossil, etc.

git is the protocol, GitHub, GitLab, BitBucket are providers

Distributed: It can be used offline without any remote server, or with multiple server

Common pitfalls

1. Not using version control :)
2. Not committing often
3. Committing with meaningless messages
4. Committing logically unrelated changes
5. Committing input datasets
6. Committing passwords!

Xkcd 1296 by Randall Munroe

Git command line

You can use git with the graphical interface integrated in IDEs, but is convenient to learn to use it from the command line.

IDEs interfaces can change, the CLI doesn't. You always have it.

It's faster and simpler, once you get used to it.

On Windows you can use git-bash

Useful commands cheatsheet

Just the essential:

- git init - start version control
- git clone - to copy a repository on your machine
- git status - show the current state of the repository
- git add - add a file or its changes to the staged commit
- git commit - snapshots the current state
- git fetch - retrieve the metadata from the remote repository
- git push/git pull - sync to/from your local repository
- git checkout - change the local branch
- git merge - merge with another branch
- git branch - create, list or delete branches

Clone a repository

git clone git@github.com:jacopofar/IntroPythonForDS.git

or

git clone https://github.com/jacopofar/IntroPythonForDS.git

the result is the same, in general using the key is better: you can later generate deploy keys to clone the repository without sending around your own secret key

See the history of a repository

git status is always useful to get the current state of the repo

git log shows the history of it

git log --all --oneline --graph to get a representation

git checkout to change current branch

Create a new repo from scratch

1. make an empty directory
2. make a file
3. initialize the repo
4. add all files
5. commit the changes

1. mkdir practice-repo
   cd practice-repo
2. git init
3. touch README.md
4. git add *
5. git commit -m 'initial commit'

Try to commit other changes, and see the history.
Also try to use **git diff** and **git diff --cached** before and after modifying and adding a file

Push the changes

You can use **git push/pull** to update a remote server with the local changes or retrieve the changes.

You can have multiple remote servers, if not specified the one called **origin** is used

If you have an account on GitHub or others you can try right now

Branching

You can create a new version of a repo and work in parallel on multiple version. For example if you want to add a feature that is not trivial

git checkout -b new-branch-name to create it

Try to branch and commit on the new branch, and see the history

Merging

You can incorporate the changes from a branch into another using merge

first go to the target branch:

git checkout master

now merge the source branch

git merge branch-name

Some people use **rebase** for this purpose, is slightly more complex and we'll skip it

On the terminology

A **commit** is a snapshot of the whole repository, not the change of it

A **branch** is a label on a commit which is moved forward automatically, not a branch of the commit tree

The way git calls things is often not very clear

Conflicts

What if a file was changed in both branches and you try to merge? You have a **conflict**

Let's try to trigger one. Notice that the branch we created previously is still there.

use **checkout** to move between master and the other branch, and commit some change in both. Try to merge and see what happens.

We have to solve them by hand and commit, git cannot infer what we want to do with the files.

.gitignore

You can ask git to not track files putting them in the .gitignore

Use it for credentials, dataset files and editor/system files that do not have to be under version control

Ideally, a file should be either in .gitignore or under version control

Use **git status** freely to check the status of the repository

Contribute to a project on Github

(Also valid on Gitlab and others)

1. **fork** the repository

Go deeper

Material from a previous year

https://github.com/ADGEfficiency/programming-resources/blob/master/git.md

A gentle introduction:
https://www.growingwiththeweb.com/2014/02/a-gentle-introduction-to-git.html

The official tutorial:
https://git-scm.com/docs/gittutorial