Software courses

Daniele Schlagenauf

June 2024

Chapter 1

Git

Exam notes The exam is formed by:

- Homework: Create an account with a repository on GitHub and perform first git push
- Multiple choice question, 30 question in google format

In all the cases, notes book etc are allowed.

Version Control System Known as VCS are system develop to **store the changes** of a file over time, eventually recall them. Can be different type

- *Local*: Not share info, everything stay on the same machine
- *Centralized*: A single copy, which everybody download and then modify before update
- *Distributed*: Each user has own copy then all sync together

1.1 Git

Git is one of the most famous VCS, do not work well with too large file, binary file (or pdf) and with sensitive data since shared. While is largely since fasst, secure, open source and scalable. Git can be use both **CLI** (Command Line Interface) or **GUI** (Graphical User Interface), the first allow a most powerful but complex approach while the second is more understandable by humans beings but limited option and not flexible as CLI.

The main structure is:

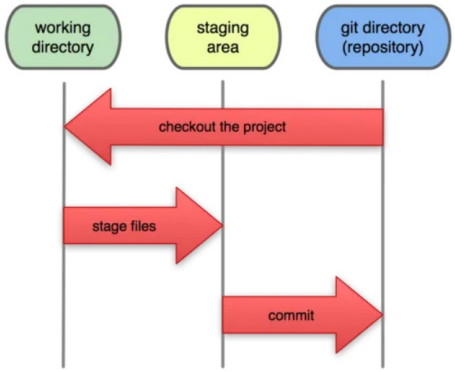

Figure 1.1: Git structure

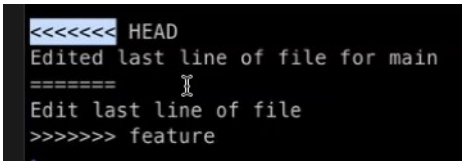

Figure 1.2: Conflict case, in merging

Where the *working directory* is the local folder where the file are stored and modified, the *staging area* is a middle step to compute before the commit abd stay on the local machine while with a commit is possible to store the new files on the *Git repository*

1.1.1 Command

The main CLI command will be show, in particular the blue ones will refer to the used of branches, for a intermediate approach while in red the more advanced ones:

- **git init**: start the filder to git
- **git add**: Command need to pass from working dir to staging area, after a space is possible to add the $< filename >$ to add, with extension, or with the . add all the changes.
- **git push**: commit the file on Github (after setting show in **??**)
- **git commit -m "msg"**: create a commit with the comment on what is change inside
- **git mv** $< old.txt >< newfile.txt >$: allow renomination of file without lose the track of the changes
- **git reset**: Allow to recover an add file (reverse of add!)
- **git diff**: compute the difference between working and stage area, should need a proper IDE to be human understandable. (–staged: diff between stage and commit)
- **git log**: Allow to visualize the list of commit done on the repository. Some option are –graph (color, use mainly for branches), –oneline (shorter versione) $-< name >$ (see the log of specific file)
- git checkout -b <branchName >: Create a new branch with the given name if not already created
- git log –graph –all: now the branch is more visible

- git push origin feature: commit only the branch not the main
- git switch <branchName >: allow to switch from the branch to the one specified
- git merge <branchName >: from the directory of the branch that should be receive the merge (main mainly), specify which branch merge
  Possible conflict: some conflict could happen when try to merge a file that is modified in both branches, vim result in **??**. The idea is work on files only in the external branch not touch the one on the internal ones and allow a single person/group on each file hence known what was modified.
- git config –global alias: create a routine, that can be run instead of a very large command
- git commit –amend: delete the last commit, use when forget a file, or commit something wrong. If add **-m "new_msg** change only the commit message, for example for type error
- git revert <commit >: delete the commit specify and commit the actual one
- git reflog: more clear log command
- git rebase -i branch
- git subase

Intermediate level To use Git at an int level, usually are done dy the use of **branches**. This allow to work better in groups project, since create a local file in the branch which will be modified without effect the main, that should be stay as clean as possible. The branches will then merge to the main once some kind of final version is reached in the branch.

Some examples shows how use cleverly the branches: the main (or master) stay clean, and create a branch to developing new part which can also be branched to develop new feature that will then merge on the branch and then in the main, as seen in **??**

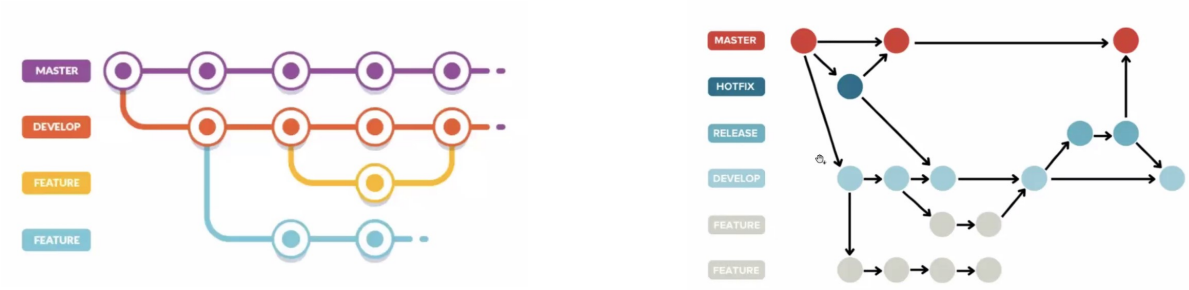

Figure 1.3: Examples of branches

When working on other project or bigger, before merge with the main project must use a **poll request**, that will be accepted by the owner of the project if approved.

.gitingore file

This particular file, create in the git folder allow to create a list of file, of type of file (like all the file with ,mw) that will **NOT** committed on GitHub. The file to not commit are usually: compiler, log, Temporary file, Configuration, system files and PDF (mainly with latex, share only .tex)

If a file is alreafy commit to delete it also from commit, before inserti in the file use the command git rm $< filename >$

1.2 GitHub

Website that allow to store commit and allow team project:

1. Create a GitHub account
2. Generate a SSH key and create a SSH-agent (link1)
3. Connect GitHub by means of the key (link2)
4. From GitHub page, create a new repository, in which after named it can also pre define
   - Readme file
   - .gitignore file
5. Create in windows a folder, the one which will commit
6. Reach from power-shell the folder, run power-shell as administrator. Here run:
   - git init: initialize the folder as git
   - git remote origin `git@github.com:<Gituser>/<GitRepository>.git`.
     As example `(i.e git@github.com:DanieleSchlagenauf99/gitCourse.git)`
   - git remote -v: to check if set properly
   - git pull origin main: pull the full image from GitHub, is needed to download the latter version, or the first time to download and update the file create (as Readme)
   - Commit once the work is done:
     - git add . : to staging all the file modified (see **??**)
     - git status : check if properly stage (should be green)
     - git commit -m "message": Create the commit and with the message in which is describe what is done
     - git push -u origin main: commit on GitHub

1.3 Powershell command

Some useful comand to known in powershell:

- Get-ChildItem -Path . -Hidden : To shows hidden element in folder
- New-Item -ItemType File -Path <> : To create a file <name>in the folder
- echo "hello world" >> file.txt: create a file.txt with the line hello world