Coursera and Google: Introduction to Git and Github

Traian Matisi

25 de outubro de 2023

Git is a CVS (control version system).
Github is a database of programmers and their codes (like a social network).

1 Cheat sheet

- git config - -global user.<*>
- git config -l
- git init
- git clone
- git remote -<?>
- git diff -u -U -C<*n> -y <file> <file> ???
- git status
- git add <file>
- git commit -m "message"
- git push origin main
- git fetch
- git pull
- -

2 Differetiating files with DIFF command from command line inteface:

- [user@pc] ~$ diff helloWorld.py helloFriend.py
  This command will show the difference between the two files.
  It's output will return only the lines that are different between the two files.
  The greater than sign (>) means the line was added to the file
  The lesser than sign (<) means the line was removed from the file
  The number codes, like (5c5, 6), means the line 5 changed to line 5 and 6
  The number codes, like (11a13, 15), means lines 13 thru 15 was added

- [user@pc] ~$ diff -u helloWorld.py helloFriend.py
  Returns the difference in a more friendly format
  + means a line was added
  - means a line was removed

- Other diffs:
  wdiff: Returns the different words highlighted
  meld:
  KDiff3
  vimdiff

- Another usefull ways to use diff, some more friendly, some less:
  –

- [user@pc] ~$ diff -u old_file.c new_file.c > change.diff
  It will generate a file containing all the changes between both
  It can show any external work by peers, and also gives us an easy patch to use with the PATCH command.

- [user@pc] ~$ patch old_file.c < change.diff
  Will insert the changes loaded in the diff file. Note the names of the files, commands and the previous example.
  Now the old file has the patch and is equal to the new file.

- But, most commonly, we'll use these and other tools and features through another tool, called VCS (version control software), like git and whatnot.

3 Configuring repositories and using git

- [user@pc] ~$ git config –global user.email "traian@tutanota.com"
  This command will set our client machine to the mail designed.

- [user@pc] ~$ git config –global user.name "traianMatisi"
  This command will set our client machine to the user name designed.

- [user@pc] ~$ cd C:/a>b>c... or /home/a/b/c... followed by...
  First we will change to the directory of our choosing with "cd /path to my folder/ "

- [user@pc] ~$ git init
  Then, we'll start the choosen folder to initialize (start) the git control

- [user@pc] ~$ git add old_file.c
  This command will track the argumented file that is in some "git init"folder

- [user@pc] ~$ git status
  Will show the mods and pending commits

- [user@pc] ~$ git commit
  Will write in the git directory files the changes made so far in the non git directories files, and we will be prompted to write a message about the commit. The files are in the working tree area, the non commited mods are in the staging area, and the commited mods are in the index (.git directories)
  These changes are made only locally in the .git directory and local files, the .git stores all the made changes (like diff files content) througout the work and the files stores the current data only.

- A good analogy to understand is: The working tree is your desk, the staging area is the document cart, and the .git is the file cabinet.

We make the mods in our desktop
~$ git add ...fileName...
Will add the list of mods to the cart/staging area.
~$ git commit -m ...type the commit message...
Will store the mods in the .git directory/file cabinet

- The basic git workflow commonly will have the following commands (we will include non git commands):

  - [user@pc]~$ mkdir ...folder_name...
    If necessary, we will create a directory to work. If not just jump to the needed step
  - [user@pc]~$ cd ...C:/ or /home...
    We will change to the created directory.
  - [user@pc]~$ git init
    We initialize the current directory as a .git repo
  - [user@pc]~$ git config -l
    Will show the config in the repo.
  - [user@pc]~$ vim hello.c *or* touch hello.c *or* chmod +x hello.c
    We'll create a file to execute, use whatever you feel confortable with.
  - [user@pc]~$ git status
    Will show the files status in the .git directory.
  - [user@pc]~$ git add hello.c
    Will start .git tracking in our file.
  - [user@pc]~$ git commit -m 'Learning the git workflow'
    Will index the staged mods in our files with the -m message.

- Remember to write good commit messages (good practices), some companies have their own rules to call a good commit message.
- [user@pc] ~$ git log
  Will display the history of the commits.
- [user@pc] ~$ git commit -a
  Adds the staged all acumulated. Dont confuse -a with add
- [user@pc] ~$ git log -p or - -stat
  To look, by scroling down and copy the commit identifier, followed by...
- [user@pc] ~$ git show q3j24987dfh23984khf...w34rd123
  To look the patch to verify it's validity.
- [user@pc] ~$ git diff
  Equivalent to the diff -u talked earlier in this document, it can receive files names as arguments to diff only the files you want. Notice that, git diff will only show the staged mods, wich was added with git add -p (-p will show changes to see if you want them all). To show non staged and staged, usen git diff –staged
- [user@pc] ~$ git rm <fileName.c>
  Will stop tracking and delete the argumented file. After, when commit, we'll need to write a message about why we deleted them (cause we'll need to commit the deletion).
- [user@pc] ~$ git mv <old name> <new name>
  Will rename/move a file, also needs a commit after it.
- [user@pc] ~$ _-_echo .DS_STORE_-_ > .gitignore
  To ignore the argumented file.

4 Undoing unwanted changes

- [user@pc] ~$ git checkout
  Discards the non staged changes, meaning it checks out the latest snapshot
  Usefull tip, -p will go through every little change you made among the two snapshots.

- [user@pc] ~$ git reset HEAD
  Will unstage the added mods. The -p will permit chosing wich mod to reset.

- [user@pc] ~$ git commit - -amend
  Will overwrite the previous commit. Locally is ok to amend, but if you're working in a public repositorie avoid it at all costs.

- [user@pc] ~$ git revert HEAD
  It contains the invert of the last commit. And a automatic commit with the old version is made.

- [user@pc] ~$ git revert q3j24987dfh23984khf...w34rd123
  Will rollback to the id 'd commit. It's like a time travel to the past.