Git & GitHub

Cheat-Sheet

Git

- **git --version:** to know the current installed version of *git*
- **git config --global user.name "user name here":** to let *git know who you are*
- **git config --global user.email "user email here":** to let *git know who you are*

Note: --global: to set the user name and email for every local repo. If you want it just for the current repo remove *--global*

- **mkdir direcctory_name**: to create a directory with name direcctory_name
- **cd direcctory_name**: to change directory to the directory with name direcctory_name
- **git init:** git creates a local repository and a hidden folder within the directory to notice the changes

Note: if you make any changes to the files in the directory, git will be aware of it and there will be 2 kinds of files:

1. ***Tracked files:*** *files that are in both the directory and repository so, any changes in them are noticed*

2. ***Untracked files:*** *files that are in the directory only so, changes in them are not noticed... need to be added to the repository through 2 steps:*

    1. *Adding the modified/new file to the staging environment*
    2. *committing the staged file to the local repository*

- **git status:** to see which files are tracked with changes in them and which are un-tracked
- **git add modified_file_name . modified_file_ext.:** to stage the modified file

Note: to stage all files in the directory use any of the following

git add . git add --all git add -A

- **git commit -m "message here"**: to add the staged files to the local repository and -m "..." is used to add a message with the commit which is a must

Notes:

- ***Staging area** is very important area **although it does not add the file to local repository** as sometimes you will have **some files that you do not want to commit** … **.gitignore***
- **Staging area** may be a waste of time **in case of small updates in tracked files**. So, you can use ***git commit -a -m "message here"*** to **stage and commit in one step**

- **git log (or git log origin/branch_name) :** to see the history of commits
- **git command_name –help:** to see all options for a command
- **git help --all:** to see all possible commands
- **git branch branch_name:** to create a branch with name branch_name
- **git branch:** to see all the current branches locally
- **git branch -r:** to see all the current branches remotely
- **git branch -a:** to see all the current branches on local and remote repositories
- **git checkout branch_name:** to move to the branch named **branch_name**
- **git checkout -b branch_name:** to create a branch of name **branch_name** and switch to it

Note: a branch is a separate copy of the local repository

- **git checkout branch_name1 then git merge branch_name2:** to merge branch2 into branch1

Note: in case of **merge conflicts** open the file , **fix the conflicts** and **check the status from time to time** till all conflicts are solved **then do a commit**

- **git branch –d branch_name:** to delete the branch named branch_name

Git and GitHub

- **git remote add origin remote_repo_url:** to link the local repo. And a remote repo. With the given url
- **git push origin branch_name:** to push what is on branch branch_name to the remote repo. on a branch of the same name
- **git push --set-upstream origin branch_name :** to push on all future pushes to a certain branch just in the future type: **git push origin**

Note: you can **push ignoring what is already pushed**, **without pulling**, using the command:

git push origin –f branch_name but **do not do it when you are working with a team** as some recent changes of someone may be lost. It is better to pull at first

- **git fetch origin(or origin/branch_name):** to get all the change history of a repo. or a branch
- **git diff origin/branch_name:** to see the difference between local and remote branch of name **branch_name** (used locally as well)
- **git merge origin/branch_name:** to merge the local and remote branches of name **branch_name**
- **git pull origin (or origin branch_name):** to fetch and merge in one step

Note:

- when the **remote repository have more branches than the local one** and you do **git pull** , **only the branches of the same name as the local branches are pulled but once you checkout to the non-local branches locally**, all their contents are pulled
- **To merge branches remotely**, do a **pull request** to **announce the changes are done** and **ask for merging the branches**, **then you can delete the merged branch**
-

- **git remote –v:** to see the remote repo.(s) you are connected to

Note: links to remote repo. **have short names** as **origin** **set on adding links** and **used in some commands** but **we can change this name** using command:

--> **git remote rename old_name new_name**

- **git remote rm remote_name:** to remove the remote URL of name remote_name

Some GitHub Features:

- **Forking:** it enables you to copy a repository to work on it as a base of your work , but since it is remote copy, you will need to get this copy with logs and versions locally. This is done via **cloning**

  --> git clone URL directory(optional … default is creating in-place directory)

  Note: cloning = init + pull … that is why **it is used at the beginning of project only**

Some Git Features:

- **.gitignore (one of the advantages of the staging environment):** used to commit only some files to the local repository from the staging area and not all of them

1. First create the **.gitinore file** : **touch .gitignore** …

   it will be added to the local repo

   Note: there may be **more than one .gitignore file distributed in subfolders**

2. **Open the file** and type your rules according to the following:

- **#** : comments
- **name** : files or folders named **name** are ignored
  - **name/:** folder of name **name** are ignored
  - **name.ext:** file of name **name** and extension **ext** are ignored
  - ***.ext**: file of extension **ext** are ignored
  - ***name**: files or folders ending with **name** are ignored
  - **name?:** files or folders ending with **name + one unknown character** are ignored
  - **name[a-z]**: files or folders ending with **name + one unknown character in specified range** are ignored

- **name[abc]**: files or folders ending with **name + one unknown character in specified set** are ignored

- **name[!abc]**: files or folders ending with **name + one unknown character not in specified set** are ignored

3. remove any **cached (staged) unwanted files**, **if existed** using:

git rm --cached file.ext

4. **stage** and **commit** all files then push them and **notice the absence of the ignored files**

- **committing previous commits again (reverting):** used in case we make a mistake in the recent changes and we want to re-commit one of the previous commits, so you have 2 options:

1. **git revert HEAD --no-edit:** to re-commit the last commit without changes even in the commit message

2. **git revert HEAD~number --no-edit:** to re-commit the commit before last commit with number commits without changes even in the commit message

- **reseting to a previous commit: unlike reverting that adds new commit** to the commit stack, **it chooses one of the previous commits** and **change its order to be at the head of the stack.** Needs the hash number of the commit, so do the following:

1. **git log –oneline:** to get the **hash no.** of each commit

2. **git reset hashNo.:** to do the reset

Note: it is better to use **reverting** when you undo **someone's bad pushed work** and **reset** when you undo **your bad not pushed work** so as not to be recognized in the history

- **amending:** to combine the staging area with the latest commit and replace the last commit

--> **git commit --amend –m "message"**