# GIT Knowhow: *(also refer to:* [*https://youtu.be/lG90LZotrpo?si=Tl3j74FzSJPuzmmB*](https://youtu.be/lG90LZotrpo?si=Tl3j74FzSJPuzmmB)*)*

## Basics

* **git init** (to create local repository)
* Create ‘.gitignore’ file with folders and files to never stage)  
  - note that you can stage only specific files under ignored folders by exclusion: !file\_name
* **git add .** (staging all files while ignoring the unwanted files, e.g. repository or virtual env)
* **git status** (to track on the staged files, modified files and active/checked-out branch)
* **git diff** (gives you a quick diff snapshot, current pre-staged vs. last commit state)
* **git diff - - staged** (more popular, gives you a quick diff snapshot, current staged vs. last commit state)
* **git reset** (if having undesired files staged, update the ‘.gitignore’ and run re-staging
* **git commit -m “first commit”**
* **git branch -M main** (renames/Modify the current active to main)
* **git branch** (to see current active branch)
* **git log** (track on commits) inwards below for more advance on this!
  + **git log [FILE\_NAME]** (track the historical commits on a specific file)
  + **git log -L ,:[FILE\_NAME]** (track the historical commits on a specific file with git diff per commit – per entire file. Note that it is case-sensitive for file/folders names).
  + **git log -L ,15:25[FILE\_NAME]** (track the historical commits on a specific file with git diff per commit – per selective file slice using lines numbering).
  + **git log -S [STRING] -p** (tracking historical commits on a specific string on all files, with git diff per commit. Git grep may not find it if was removed from the code base at a certain point).
  + **git grep [STRING]** (return only the lines where the exact phrase, and not as subset, is found).
* **git remote add origin https://github.com/shai-baranes/Slot\_machine.git**  
  (get link from GitHub for the first commit only or for connecting new computer…)
* **git push -u origin main** (also for first commit)
* **git push -u origin from\_perplexity** (and if we wanted to update a certain branch out of many to be in focus in our local repo).
* *Note that git creates the first branch as ‘master’, yet we can decide to change it to ‘main’ and we shall have no history for ‘master’.*
* *If all branches are stored locally, no need to pull from repository to get to an earlier branch/code.*

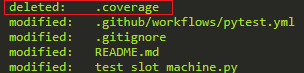
## Advance

* **git log --all --decorate --oneline --graph** To display a commit per line, e.g:   
   *(also depicting visually the tree branches – can be in Alias)*  
  ----------

\* **8f5a062** (HEAD -> main, origin/main) updating the README file

\* **2f4c8be** adding README.md

\* **faad10c** first commit   
----------

* **git reflog** (another way to see all with head [i] for timeline tracking – not replacing the above)
* **gitk** another lovely way to get a visual representation of all commits and branches!
* **git branch side\_branch** ; **git branch for\_testing** (adding 2 more branches)
* **git branch** (to see all branches)
* **git checkout side\_branch** (HEAD pointer is now moved to the new branch)
* **git checkout -b side\_branch** (for both creating and checking out into the new branch)
* **git rebase -i HEAD~4** (checkout back in time 4 commits – from where we are now; it is possible to replace HEAD w/ main or master – also enables to delete only partially).
* **git reset –hard HEAD@{5}** (to undo the above rebase cmd, helpful to track rebase changes by running: **git reflog**; note that **git reset –hard HEAD@{1}** will now undo the undo 😊)
* **git commit -a -m “commit info”** (to both ‘stage’ and ‘commit’ any modified file)
* **To merge 2 diff branches , both stemmed from the master/main:**
  + **git checkout master** (move pointer/header back to master)
  + **git diff master..side\_branch** (shows what will change when merge side to master)
  + **git merge master\_branch**   
    (from the master we merge the side\_branch, ff merge since having direct path)
  + **git branch --merged** (shows which branches merged to the branch we’re on)
  + **git branch -d side\_branch**   
     (after the above cmd we can safely delete the ‘side\_branch’)
  + **git branch -D side\_branch** (forcibly deleting unmerged branch. try to avoid.)
  + **git push --all origin** (to push all local branches to the remote git-hub repository)
* **Working with a clone from remote git repo:**
  + **git clone** [**https://github.com/shai-baranes/MISC.git**](https://github.com/shai-baranes/MISC.git)*(taken from ‘Code’ in repo)*(get you repo to other user or other folder of yours…)
  + **git fetch origin** (if during activity, you want to check whether a new change committed since).
  + **git status** now tells you how behind are you; e.g.   
    *Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded. (use "git pull" to update your local branch)*
  + **git diff origin/main** (quickly see changes. *if your main is ‘master’ then master it is)*
  + **git pull** to pull these new changes(no need to commit)
  + **git stash --include-untracked** (if you pull and wants to ignore local non committed changes – note you can later find it or discard it) – now try **git pull** again…
  + **git reset --hard**  (discarding local changes - such as stashed)
  + **git clean -fd** (rarely for cases you also want to clean the untracked/ new files. Irreversible!)
* **To delete a cached file that is not ignored although listed (since already in repository):**
  + **git rm -r --cached .**
  + **git add .** (you get something similar like the below output)  
    
* **what to do in case I am not satisfied on my last changes (revert)?**
  + **git reset --merge** (if there are conflicts and I’d like to revert to previous state – instead of changing files, stage the change changes and commit resolution).
  + **git reset --hard HEAD** (force action to discard all changes since last commit).
  + **git checkout main [or master]** (to move to my main branch before killing the unwanted branch).
  + **git clean -n** (safe step: If still have untracked files, this cmd will clean it – vs.)
  + **git clean -f** (permanent step: If still have untracked files, this cmd will clean it).
  + **git branch -d <branch-name>** (if changes are merged)or **git branch -D <branch-name>** (if changes are not merged and I don’t really care).
  + **git push origin --delete <branch-name>** (to delete the undesired branch from the remote/github repository).
  + **git fetch -p** (recommended after deleting the remote branch to avoid references to remote branch that is no longer exist).

## Aliases

* **git config --global alias.last “log -1 HEAD”** (example for basic alias for logging the last commit; execute by: >> **git last**)  
  ***\*note: in my case, the ‘single quote’ for the alias was not working!***
* **git config --get alias.[YOUR\_ALIAS]** (for inspecting a specific alias, wheter exists and what aimed to do?)
* **git config --get-regexp ^alias** (listing your user defined aliases – entire list)

## Extras

*(****lovely tips from****: https://www.youtube.com/watch?v=aolI\_Rz0ZqY&t=437s)*

* **git blame [file\_name]** (line by lines overview on a file while adding description on the non-committed lines).
* **git blame -L 15,26 [file\_name]** (same as above, with a slicing option for the lines we want to focus on -> reminds the functionality of ‘**git diff [file\_name]’**.md by with the option to widen the FOV).
* **git blame -w -C -C -C -L 15,26 [FILE\_NAME]** (each line has the developer who wrote it and the one to blame as needed – for muti-users project. Also ignoring white spaces and allegedly function/code movement if not affecting the functionality).
* **git config --global rere.enabled true** (if you rebase and resolve conflicts, it learns how you resolve and do same in future conflict)
* **git maintenance start** (adds periodically crone job with maintenance task on my repository. All will run faster going forward).

**TBD – check how to push the entire history with all sub-branches…**

**TBD – learn more about pull / deploy / clone?**