Setting github user

* **git config --global user.name "Mona Lisa"**
* **git config --global user.email "your\_email@example.com"**

:q exitgit

* Create repo
* **git init** // create repository
* **git add (name or . )** // add to stage, stage means they are traced. This will create a master branch
* **git add A**

Commits

* **git rm (--cached) (name or -r . )** // remove from stage
* ??git restore --staged // version 2.3
* **git commit -a -m "message"** // commit add message. **Message is in double quote**
* **git checkout -- (name of file) .** // go back to before changes in the work tree and clean it
* **git reset HEAD .** // unstage files and bring them to the work tree
* **git reset --hard reposid** //delete created files after that stage from the repository
* **git show (HEAD commitid or tagname)** //show changes in that commit. **Shift + q** to exit.
* **git log -p** // show change in every file
* **git log -p -- path/to/file // show the changes of one file**
* **git log -number --oneline** // see commits the last number of commits, oneline-> short
* **git diff (HEAD or --staged)** // see changes of current work compared to the last commit or last added stage

Branches to keep changes isolated

* **git branch name**
* **git branch -a** // list of branches
* **git checkout** **name** // change branch to name, if changes not committed in the name they will appear in master after changing branch to master waiting to be committed
* **git branch -d name** // delete branch when you are on another branch and you're done with it
* **git merge branchname** // merge changes of branch name to the current active branch
* **git log --graph** // show commits in graph
* **git checkout -b nameofnewbranch master** //create a new branch and switch to it

Stash

used for branches having changes but we don't want to commit them and we don't want to see them when we change to another branch, When you stash they no longer appear in the status

* **git stash** // makes the changes go to stash and we see files before changes of the last commit, message is the last commit
* **git stash list**
* git stash drop stashname // deletes stash
* **git stash save "message"**
* **git stash show -p stashname**
* **git stash apply stashname** // brings that stash changes to the work tree and shows in status as must be committed app.ssbut does not remove from stash
* **git stash pop stash name** // does the above with removing from stash space

Tag

is used for tagging your releases and creating versions the difference with branches is that if you go to that tag and if you change something it will not affect it you have to create a branch and do your changes because it is just a tag

* **git tag -a v2.0 -m ""** // for pointing to a specific time and for versions, -a annotate
* **git push origin (tagname or --tags)** //simple push don't push tags, you have to use this way
* **git push --force-with-lease**
* **git checkout tagname** // to go to that tag

Ignore

is used when we don't want git to track certain files for example npm in node.js or vendor folder in composer in laravel for the package manager

* **touch gitingnore** //to add gitignore to git
* open the git ignore file created in the project and add files and folders =>foldername/ , \*.txt, !name and … (<https://git-scm.com/docs/gitignore>)

blame

is used to see who has changed a line

* **git blame filename -lnumber, number**

bisect (binary search commit)

to find when a bug is created, it searches commits in binary search and you say when the bug didn't exist

* **git bisect start**
* **git bisect bad**
* **git bisect good commitid**

gpg

git uses this to sign things like tags, either no one signs or all teams should sign on all commits, You can make it a must to sign

* **gpg --list-keys**
* **gpg --gen-key**
* **gpg --list-secret-keys --keyid-format lONG** // show my hidden keys
* **git config --global user.siginingkey keyfromabovelsit** // to assign a created key to the user
* **git tag -s tagname -m ""** // to sign tag
* **git show tag**
* **git tag -v tagname** // verify tag
* **git commit -S -m ""** // to sign a commit

Remote (Github)

* **git remote add nameofremote addressofrepongithub**
* **git remote rm <remote-name>**
* **git remote (-v)** // list of remotes (verbose)
* **git remote remove remotename** // When we don't what is a remote or we don't need it
* **git push (-u nameofremote nameofbranch)** // put rep to GitHub (for the first time)
* **git pull remotename**
* **git clone url nameoffolder** // git clone project in local
* **github desktop program** // name of commit - resolves //1 (issue number 1) and then pushes to the origin
* **fork, pull request** // to create a copy in our account
* **README.md** // always use to explain your work, sharp sign h1, Markdown is a lightweight markup language for creating formatted text using a plain-text editor for styling with GitHub markdown
* issues, push or merge based on who fixes it
* license.md //conditions and rules we should follow, can be expressed based on specific template like MIT, <https://shields.io/> and badge.fury.io icons for readme for better understanding of your work
* topics in GitHub //tags for your code to appear in the Github search
* How to push in a company rep: we create a remote to an HTTP or HTTPS or ssh of companies git or GitLab or an address and connect to it by key, user/pass or ...

GitLab

thedifference with GitHub is that we can install this on a local machine, is used for private works, and you can install it for company private git

you have to install and config it yourself or you can use the virtual machine (GitLab VM) bitnami.com/stack/gitlab/virtual-machine

in companies, they give us access to a remote which we can push to it

lfs, Upload Big files to GitHub

* **LFS**: is included in the windows package but it should be installed on other OSs.
* git lfs track "\*.psd" // track the big files, and then add and commit and push
* commit then push