**BACKGROUND**

There are three statuses:

* Committed: in our databases
* Modified: Changes are not still affected in our databases.
* Staged: Ready to be confirmed.

There are 3 types of version systems:

* Local Computer.
* Centralized. Relies in a super server (repository) where data is storage.
* Distributed control system (git): each participant has its own copy of the project.

There are 3 states

* Working Directory
* Stage Area.
* Git Directory (repository)

**CONFIGURE GIT:**

git config –global user.email [abc@gmail.com](mailto:abc@gmail.com)

git config –global usen.name “Misael”

git config –global color.ui true

**TERMINAL COMMANDS**

git--version**:** gets git version

cd: move within folders

mkdir : create folders

ls: show folder

clear: clear terminal

touch: create files

rm: delete files

rm –rf folder: delete folders

**CREATE/INITIALIZE REPOSITORY**

git init *nameRepositor***:** we can either specify or not the folder

**GIT COMMANDS**

git status**:** show directory status

git add *file.ext*: Add files to **staging** area

-A: add all files

-n file.ext: simulate adding the file

git rm --cached file.ext: unstage file from stagin to working

-f file.ext: delete completely files

git commit -m “description”: commit changes

git commit --amed -m “description”: concatenate the last commit with the new adding(git add), to confirm the last message: esc, :, wq

**TAGS:**

git tag 0.5**:** light way

git tag 0.3 20dc755eae63639632ee995a386d3ff789e4f7ea**:** light tag a previous version

git tag -a 0.5 -m ‘stabled version’**:** annotated

git tag -l**:** shows the tag list

git tag -d 1.0: delete tag

git tag -f -a 0.1 -m ‘Initializing the project’ 20dc755eae63639632ee995a386d3ff789e4f7ea**:** rename tag

**LOGS**

git log**:** default log

git log --oneline**:**

git log --online --graph**:** showing bifurcations

git log -3: last 3 commits

**DIFFERENCES BETWEEN VERSIONS**

git diff 20dc755**:** compares the specified commit and the previous commit

git diff 40fcd35 20dc755**:** compares the specified commits

**RESET: OVERWRITE SOMETHING**

git reset --soft 20dc755: remove the last commit (files are in staging area ready to commit) we have to choose a SHA previous to the change we want to remove.

git reset --mixed 20dc755: remove the last commit (files are in working directory now).

git reset --hard 20dc755: remove completely the last or specified commit, we can go back to the past or forward to the future

**CONFIGURING A DIFFERENT TEXT EDITOR**

git config --global core.editor "sublime --wait"

git config --global core.editor "'c:/program files/sublime text 3/subl.exe' -w"

**BRANCHES**

git branch responsive: create a new branch

git branch -l: list branches

git branch -d *name*: delete branch

git branch -D *name*: force delete branch

git branch -m *name* *newname*: rename branches

git checkout branchname/commit**:** move within branches or commits

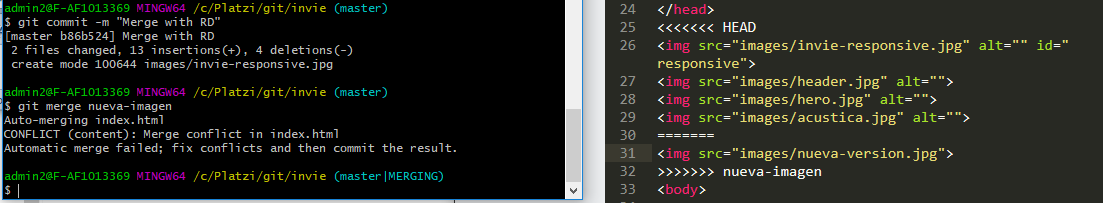
git checkout -b *namebranch*: create and switch to brand

git checkout -- *filename***:** undo de changes of a file

**MERGE**

**git merge [branch]:** It allows to merge changes in the branch specified into the branch we are in.

* fast-forward: it merges automatically
* recursive/auto-merging: both branches emerge from at the same time the merge is recursive.
* manual merge: We have to specify git which changes we want to merge, we have to edit it manually.



**REBASE.** It’s like a merge but changes the story of the project and creates only one line(without branches)

**Example**

Create two branches (UpdateA, updateB), and create a file in each branch, return to master and merge to UpdateA

git checkout -b Update A

git touch UpdateA

git add -A

git commit -m "Adding UpdateA"

git checoout master

git checkout -b update B

git touch UpdateB

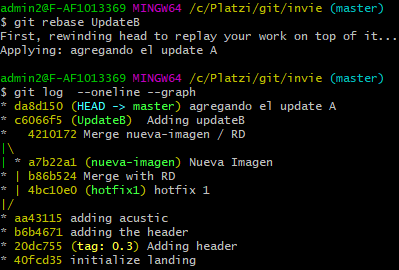
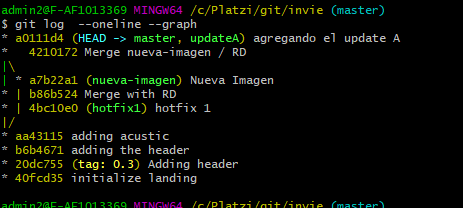
git add -B

git commit -m "Adding UpdateB"

git checkout master

git merge UpdateA (fast forward)

git rebase UpdateB



**STASH (save changes temporarily)**

git stash: In order to add changes, they must be in the staging area.

git stash list: Shows stash list.

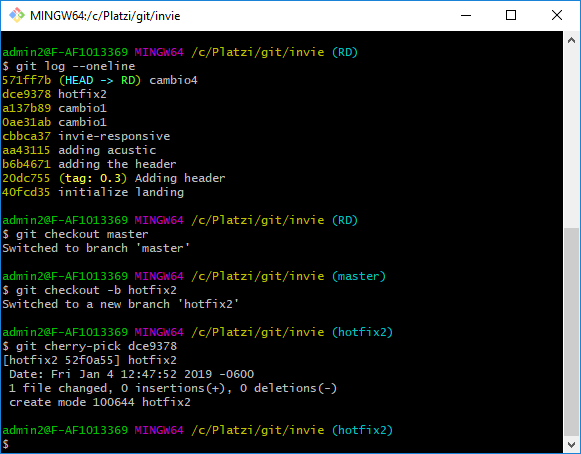
git stash drop stash@{number}**:** Deletes an stash.

git stash apply: Applies last stash

**CHERRY PICK**

Place in the branch we want to insert the change

git cherry-pick *dce9378*



**GITHUB**

**Clone/fork**

**Fork:** copy the repository in our own github without making changes in the original project

**Clone**: creates a copy in our pc

git clone *https://github.com/m1zion/invie-git.git*

**ADDING A SSH KEY TO OUR REPOSITORY:** this allows us to communicate with github

ssh-keygen -t rsa -b 4096 -C "misaelzion@gmail.com"**:** creates a new ssh key

pbcopy < ~/.ssh/id\_rsa.pub(MAC)

cat ~/.ssh/id\_rsa.pub(Windows)**:** copies the ssh key or we can go to help menu in **GIT GUI**

**REMOTE:** make the connection between our local repository and github

git remote add origin *https://github.com/m1zion/invie-git.git*

**FETCH**

git fetch origin master: brings changes from github to master and creates a new branch (origin/master)

git merge origin/master --allow-unrelated-histories**:** merges the changes in master

**PULL**

git pull origin master:brings changes without making a new branch

**PUSH**

git push origin master

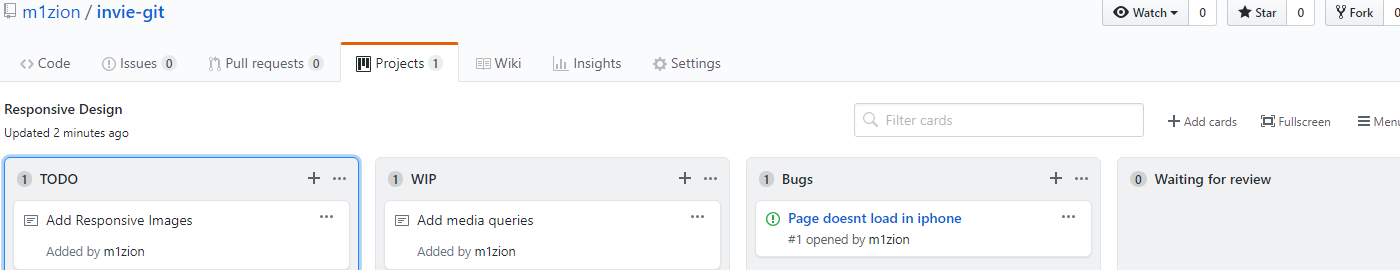
git push origin --tags

git push origin [amother branch]

**GITHUB**

**PROJECTS**

We can create a project for each big change in the repository, wa can add columns and tasks for each one, then, when a task is completed we can move it to a different column (For example “Done” column)

****

**TEMPLATE ISSUES**

We can create an issue template just by creating a new file named **issue\_template.md**, and now every time we create a new issue, the template will appear. Make sure to commit changes

**TEMPLATE PULL REQUEST**

Create a fine named pull\_request\_template.md

**IGNORE**

Ignores files when adding files, the file created has to be names as ***.gitignore*** and we edit it, listing the files to ignore.