// Changer l'editeur

git config --global core.editor emacs

// lister la config

git config --list

// Cloner sous un nom

git clone https://github.com/libgit2/libgit2 mylibgit

// Affichage courte de status

git status -s

// staging the file content

git add

// interactive staging

Git add -–interactive (or –i)

//compare l'espace de travail avec staging

git diff

// compare ce qui est stagé avec ce qui est commité

git diff --staged(or --cached)

// remove and stage removal file

git rm test

// remove file from staged area

git rm --cache file

// show commit history

git log

// shows the difference introduced in each commit

git log -p

// limits to a number of entries

git log -3

// change the format of the output in oneline per commit(short,full,fuller)

git log --pretty=oneline

// To specify the format use format option

git log --pretty=format:"%h - %an, %ar : %s"

// Limiting Log Output by date(since,after,until,before)

git log --since=2.weeks

// Limiting log output by searching string "-S". Only show commits adding or removing code matching the string

git log -Sstring

// amend a commit

git commit --amend

// unstage staged changes

git reset HEAD <file>

// discard changes in working directory

git checkout -- <file>

// list the shortnames of each remote (-v shows you the URLs that Git has stored for the shortname to be used when reading and writing to that remote)

git remote

// show more information about a particular remote

git remote show shortname

// add remote repository

git remote shortname url

// git fetch command only downloads the data to local repository – it doesn’t automatically merge it with any of work or modify what are currently in working

git fetch remoteshortname

// fetch from all remotes

git fetch --all

//git pull command automatically fetch and then merge the remote branch into local branch.

git pull remoteshortname

// list tags with a particular pattern

git tag -l "v1.8.5\*"

// create annotated tag

git tag -a v1.4 -m "my version 1.4"

// create lightweight tag

git tag v1.4-lw

// create a tag from commit

git tag -a v1.2 9fceb02

// to share tag

git push origin [tagname]

// Tag can't be checkouted but we crate a new branch at a specific tag

tag checkout -b [branchname] [tagname]

// create aliases

git config --global alias.co checkout

git config --global alias.unstage 'reset HEAD --'

// create branch - only creates branch, does not switch to the new created branch

git branch testing

// delete a branch

git branch -d hotfix

//lists commits and shows current branch (that the HEAD point to)

git log --oneline --decorate

// switch to an existing branch(SWITCHING BRANCHES CHANGES FILES IN WORKING DIRECTORY)

git checkout testing

// create new branch and switch to it at the same time

git checkout -b iss53

// to merge branch1 into master

git checkout master

git merge branch1

// Listing branchs. the \* character prefixes the branch that you currently have checked out(i.e., the branch that HEAD points to)

git branch

// Listing branches including what each branch is tracking and if your local branch is ahead, behind or both.

git branch -vv

// To see the last commit on each branch

git branch -v

//The useful --merged and --no-merged options filters branch list to that you have or have not yet merged into the branch you’re currently on

git branch --merged(or --no-merged)

// to share a branch

git push <remote> <branch> (example git push origin serverfix)

// to work on new created branch in server

git fetch origin

git checkout -b [branch] [remotename]/[branch]

// a shortcut for last command

git checkout --track [remotename]/[branch]

// other shortcut but only If the branch name to checkout (a) doesn’t exist and

// (b) exactly matches a name on only one remote, Git will create a tracking branch for you:

git checkout branchName

// to change the upstream branch you’re tracking use -u or --set-upstream-to option

git branch -u origin/serverfix

// to delete a remote branch

git push origin --delete serverfix

// With the rebase command, you can take all the changes that were committed on one branch and replay them on another one.

// It works by going to the common ancestor of the two branches (the one

// you’re on and the one you’re rebasing onto), getting the diff introduced by each

// commit of the branch you’re on, saving those diffs to temporary files, resetting

// the current branch to the same commit as the branch you are rebasing onto,

// and finally applying each change in turn.

Example :

Assume the following history exists and the current branch is "topic":

A---B---C topic

/

D---E---F---G master

From this point, the result of either of the following commands:

git rebase master

git rebase master topic

would be:

A'--B'--C' topic

/

D---E---F---G master

// Use of –-onto option

Here is how you would transplant a topic branch based on one branch to another, to pretend that you forked the topic branch from the latter branch, using rebase --onto.

First let’s assume your topic is based on branch next. For example, a feature developed in topic depends on some functionality which is found in next.

o---o---o---o---o master

\

o---o---o---o---o next

\

o---o---o topic

We want to make topic forked from branch master; for example, because the functionality on which topic depends was merged into the more stable master branch. We want our tree to look like this:

o---o---o---o---o master

| \

| o'--o'--o' topic

\

o---o---o---o---o next

We can get this using the following command:

git rebase --onto master next topic