Git cheatsheet locally

Convert a normal folder to local git repository:

git init

Work with branches locally:

git branch	Show available brances in local repo
git branch jake	Creates a branch with 'jake' as name
git branch -d jake	Deletes a branch whose name is 'jake'*
git checkout jake	Switch branch to 'jake'**
git checkout -b jake	Creates and switches branch at once.
git merge jake	Merges jake branch into the current
git diff jake	Shows differences between jake and current
git rebase jake	merges branch jake to current by putting all commits before the current branche's commits.

* If you want to delete a branch then you MUST not stading inside this branch when you are trying to delete it. Thus the most common way is to switch to master and delete the branch from there. If a branch has not been fully merged into another branch (master for example) and you try to delete it, git is going to complain and say it to you that the branch you want to delete has not fullfilled it's purpose yet. If you want to delete it regardless then you type the same command, but with capital D in this case ex. git branch -D jake.

** Imagine for example that you are working on your current branch, you have made already some changes and suddenly a colleague says to you that he is about to push to the master. You want to go to master and fetch these changes there before you proceed on your branch. If you have unstaged files git is going to prevent you from switching branches before you actually add and commit those or reset their progress. In general you can't abandon a branch when it's state is not clean (branch clean, nothing to commit).

Working with files locally:

git add jake.html | adds jake.html to the staging area

git add jake.html jake.css | adds multiple file to staging

git add --all | add all changed files to staging area

git commit -m 'something' | Creates a commit with message

git commit -am 'something' | add all and commit at once

git commit --amend* | Creates new commit, deletes the previous

git status | Checks the whole status of your git repo locally

git log | Shows all the previous commits

git log --oneline | Log the commits line by line

git log author="Kostas Diakogiannis" | Show kostas commits

git log --oneline -3 | Show the last 3 commits line by line

git diff | See the differences between last commit and now

git stash**| Save temporarily untracked files without commit

git stash pop | Bring the stashed files back to life

* It should be never used on master branch and especially after you have pushed on remote master branch. It changes history by deleting the last commit and adding a new one with all the old and new changes from these 2 commits together. (Like when you forget to add something to your cv, but in reality it deletes the previous cv, creates a new one, puts everything you had written before and the one that you are about to add). Heavily used when working locally on your own branch, and you have forgotten to do a small task with the last commit.

** Bear in mind that you are allowed to perform only one stash on your current commit state. In case you put a new stash at the same commit without popping the previous one, this latter stash will overwritte the previous one.

Connect to remote server:

git remote -v | Check origin's address (either ssh or https)

git remote add origin someUrl | Sets up someUrl as remote repo

git remote set-url origin someUrl | Changes remote url

git fetch | Fetches everything but does not merges yet *

git merge origin/master | Merges the origin/master (see above)

git pull origin master | Fetch and merge from remote master

git pull origin jake | Pull from remote jake branch

git push origin master | Push code to remote master

git push origin jake | push to remote jake branch git push --set-upstream origin master | Sets the remote master on track with the current local branch where this command is being written. After this you don't have to specify any url neither when pushing nor pulling (just git push or git pull will do the job).

* Git fetch and git merge from origin are two very well connected commands. When you fetch from origin a new layer in between remote and local is temporarily created from git. This temporary repository represents the remote/branch and is called 'origin/master' (if we are fetching from master branch of remote etc). In order to accept this whole layer and merge it to our local git repository we type git merge origin/master. This install the changes from the remote master to our local git repo and deleted the temporary layer in between once and for all.

Important note: If you don't want to mess with this stuff, try git pull instead, which is a combination of the two afforementioned commands.

Time Travelling - Undoing Changes:

git revert HEAD | Undo last commit, (git's way to ctrl + z)

git reset --HARD #7eb4eec | Go back to the specified commit

git reset HEAD jake.html | Unstage jake.html

git checkout -- jake.html | Discard changes to jake.html *

git cherry-pick #7eb4eec | Brings the a specified commit to the top of the current branch and puts the HEAD on it.

* That works only if jake.html changes have not been added to staging area yet. For unstaging see git reset HEAD jake.html above.