GIT reference

A short reference to most useful git commands

Basic

Files can have 4 status: Untracked, Unmodified, Modified, Staged
HEAD is the pointer to the current commit (branch)

`git config --list`

check your configuration settings

`git config --help`

to show all commands about configuration (--help works with any other command)

`git init`

creates a new subdirectory named .git so setup a new git repository skeleton

`git clone <url>`

creates a directory named as the project, initializes a .git directory inside it, pulls down all the data for that repository, and checks out a working copy of the latest version

`git clone -b <branch> --single-branch <url>`

clone a single branch of a repository

`git add <filename>`

To stage a file (will be included in next commit)

`git mv <old> <new>`

`git rm <filename>`

rename/delete files

`git commit -m "your comment to the commit"`

To commit the stage (create a new commit with the stage)

`git status --short`

show git status (--short has more compact view)

`git diff <file>`

shows only changes that are still unstaged

`git diff --staged <file>`

compares your staged changes to your last commit

`git log --oneline --decorate --graph --all`

print out the history of your commits, showing where your branch pointers are and how your history has diverged

---

Undo

`git commit -m 'some message'`

`git add <forgotten_file>`

`git commit --amend`

Add the forgotten_file to last commit without creating a new one (if you commit and then realize you forgot something)

`git reset HEAD <filename>`

unstage a file (remove from last, still not pushed, commit)

`git checkout -- <filename>`

To reset a file to last commit (delete you local changes)

`git revert HEAD`

creates a new commit wich does not contains the last one (undo the last commit)

`git revert <commit>`

creates a new commit wich does not contains the specified one (undo that one)

Remote

You you have setup only one remote the <remote> in commands can be omitted 'origin' is the default remote name assigned when you clone

`git remote -v`

shows which remote servers you have configured

`git remote add <name> <url>`

add a new remote

`git remote rename <old> <new>`

`git remote remove <remote>`

rename and remove remote

`git fetch <remote>`

downloads the data to your local repository. Note that fetch doesn't merge, so doesn't change you local files

`git pull`

fetch and merge

`git push <remote> <branch>`

update remote repository with your local branch This command works onlyif nobody has pushed in the meantime. Otherwise you'll have to fetch their work first and incorporate it into yours before

`git remote show <remote>`

to see more information about a particular remote

`git ls-remote <remote>`

list references (branches, tags, etc) in remote repository

`git push <remote> --delete <branch>`

delete a branch from remote

Tags

Two types of tags: lightweight (just a pointer) and annotated (with infos)

`git tag`

list tags

`git tag -a <tag> -m "some comment" [commit_checksum]`

Create an annotated tag (without -a create a a lighweight tag), with commit_checksum (or part of) tag is created referred to a particular commit (so you can tag a previous commit)

`git show <tag>`

see the tag data

`git push origin <tag>`

push tags to a shared server. Note that by default git push command doesn't transfer tags to remote servers

`git checkout <tag>`

view the versions of files a tag is pointing to NOTE: you'll enter in a 'detached HEAD' state

`git checkout master`
`will revert you to previous state`

Branches

master is the default branch. HEAD is the pointer to the current branch. Local branches aren't automatically synchronized to the remotes branches so needs to be explicitly push

Until you don't make changes to a branch it is simply a pointer to an existing commit. After a new commit is made on that branch history diverged

`git branch -v`

list branches and some infos. The '*' in the output points to the current branch

`git branch <branch>`

This creates a new branch (pointer) to the same commit you're currently on

`git checkout <branch>`

This moves HEAD to point to the testing branch (switch to another branch). NOTE: It's best to have a clean working state when you switch branches (empty stage)

`git merge <branch>`

merge current branch (HEAD). Git takes care of existing history divergence if any. NOTE: the current branch is the one modified by the merge

`git branch --merged`

see which branches are already merged into the current branch

`git branch --no-merged`

see all the branches that contain work you haven't yet merged in

`git branch -d <branch>`

delete a branch (use -D to force deletion of unmerged branches)

`git rebase <branch>`

take all the changes that were committed on branch and replay them on current one. This will create a new commit ahead pointer by the current branch. WARN: do not rebase shared branches. Rebase only of branches are local

Conflicts

When merge (both remotes and branches) you can have conflicts and merge fails. Your conflict files will then contains some notes like

```
<<<<<<< HEAD:filename
some code here
=======
some other code here
>>>>>>> branch:filename
```

you might resolve the conflict by replacing the entire block and staging file

---