Summary of Git commands

by George Creekbed: github.com/George-Creekbed

1 Main commands

- Create a new subdirectory named *.git* in project folder (here, *c:/go/to/directory/of/project*) [2 steps]:
  1. `cd c:/go/to/directory/of/project`
  2. `git init`
- Add a file, or a series of files, to version control [2 steps]:
  1. `git add *.cpp` adds all *.cpp* files in project folder
     or
     `git add singlefile.boh`
  2. `git commit -m 'initial project version'` creates initial state of repository, and adds a brief explicative comment
- Clone an existing remote repository, and set the new local one to track the remote:
  `git clone https://github.com/your-github-username/etc name-of-repository-on-local-machine` (last term optional)

  or, if remote is not on *Github*

  `git clone git://` or `user@server:path/to/repository`
- Add modified file to staged area:
  `git add file.cpp`
- Commit changes (current state becomes staged):
  `git commit -m "Your commit message here"`
- Check status (unmodified/modified/staged commits):
  `git status` verbose
  or
  `git status -s` or `git status --short` succinct

- Add a remote repository (*your-repository*) and gives it a shortname (*chosen-shortname*):
  `git remote add chosen-shortname https://github.com/your-repository`
- Copy a remote repository previously added to local. The remote is identified by a shortname (here *my-remote*):
  `git fetch my-remote`
  `git pull my-remote` copies the remote and tries to merge it with local
- Push local branch to the remote server:
  `git push name-of-server local-branch`
- List of local branches:
  `git branch`
- Create a new branch:
  `git branch new-branch-name`
- Switch to an existing branch:
  `git checkout my-other-branch` the *HEAD* pointer now points to *my-other-branch*'s current commit
- Create + switch to new branch:
  `git checkout -b new-branch-name`
- Delete branch:
  `git branch -d some-branch`
- Merge a branch into another:
  `git merge some-branch` merges *some-branch* into the current branch pointed by *HEAD*. But first go to the branch you want to merge into by `git checkout my-branch`, and only then `git merge some-branch` into it
- Synchronise local database with remote server (update local database with remote data and pointers):
  `git fetch name-of-server` to merge it with local database, `git merge name-of--server/branch-to-merge`. There has to exist a local branch by the name of *branch-to-merge* to merge the remote *branch-to-merge* into
  `git pull name-of-server` = `fetch` + `merge` operations

2 Other useful commands

- Check what has been changed but not staged:
  `git diff`
- Check what has been already staged (the two commands given are identical):
  `git diff --staged`
  `git diff --cached`
- Create a *.gitignore* file, listing files ending in some specific way as to be ignored:
  `cat .gitignore *.boh *.[oa]` (ignore any .o or .a file) `doc/**/*.pdf` (ignore any pdf in subdirs) `doc/` (ignore anything in subdir doc)
- More options to `commit`:

`git commit`	
`git commit -v`	more details on changes committed
`git commit -a`	skips staging area for modified files (no `git add` command required)
`git commit --amend`	replaces previous `commit` with new `commit` (command issued following minor corrections to already staged-and--committed files)

- Remove a file from the git project (manually removing the file from the working directory will not remove it from the unstaged area). Deletion will happen next time a `commit` is run:

`git rm name-of-file`	if unstaged. Removes *name-of-file* from working directory
`git rm -f name-of-file`	forces removal if already staged. Removes *name-of-file* from working directory
`git rm --cached name-of-file`	removes the file anywhere from git, but keeps it in your working directory
`git rm log/{}*.log`	removes all *.log* files in directory *log/*
`git rm {}*~`	removes all files ending with ~ in the project

- Rename a file in git (renaming will happen after `commit`):
  `git mv old-name new-name`
- Check contents of a file inside git project:
  `git cat-file -p [`prefix code of directory`]checksum-of-file`
  for instance, in */git/projects/0a*, open *3b69012c...* as *0a3b69012c...*
- Review the commit history:

`git log`	
`git log -p(--patch)`	displays the outut of diff between each commit in history log so you can see changes
`git log --stat`	gives summary of file changed for each commit and in total
`git log --pretty= oneline, short, full, fuller, format:"%h - %an, %ar : %s"`	various options to customize, with output format: `%h`: abbreviated hash, `%an`: author's name, `%ar`: date of change relative to now, `%s`: description
`git log --graph`	crude graphical representation of commits in their branches
`git log --decorate`	shows which commit the *HEAD* pointer points to
`git log --oneline --decorate --graph --all`	shows graphically the branch history, and where HEAD is at now

- Limit the `log` output to certain entries:

`git log -2,3,4...`	shows only last 2,3,4.. commits
`git log --since=2.weeks` or `git log --until="2008-1-25"` or `git log --until="2 years 1 day 3 minutes ago"`	shows commits made since a date, or until a date
`git log --author= 'Donald Duck'`	includes only commits by Donald Duck
`git log --grep= "a comment"`	includes only commits with *a comment* in their commit description
`git log -S function-name`	includes only commits changing the code with function-name in it
`git log --[use-any-of-the-options-above] -- t/`	includes only commits in directory path `t/`

`git log --no-merges`	excludes commits that are only merges (and are usually not very informative)

- Unstage a staged file:
  `git reset HEAD name-of-file`
- Unmodify a modified file (dangerous command, overwrites all modifications):
  `git checkout -- name-of-file`
- Consult the list of remote servers you can work with from the current local folder:

`git remote`	
`git remote -v`	shows URLs of remotes
`git remote show shortname-of-remote`	verbose

- Rename a remote:
  `git remote rename old-name new-name`
- Remove a remote repository from local machine:
  `git remote remove remote-name`
  `git remote rm remote-name`
- Tags:

`git tag`	lists tags
`git tag --list` (or `-l`)	lists tags
`git tag -l "v1.85*"`	searches and shows all tags beginning with *v1.85*
`git tag -a v1.4 -m "My annotation to the tag"`	adds an annotated tag
`git tag v1.4`	adds a lightweight tag
`git show v1.4`	shows tag's contents
`git push origin v1.4`	pushes tag to remote repository
`git push origin --tags`	pushes all local tags not yet on remote repository
`git tag -d v1.4`	deletes tag locally
`git push origin :refs/tags/v1.4`	deletes tag from remote repository by updating it

- Create macros (aliases) to commands:
  `git config --global alias.ci commit` now can commit by typing *git ci*
- Create custom commands:

`git config --global alias.unstage 'reset HEAD --'`	joins `git unstage filename` with `git reset HEAD -- filename`
`git config --global alias.last 'log -1 HEAD'`	common custom command, used to show only last commit

- More options to list local branches:

`git branch -v`	shows last commit for each branch
`git branch --merged`	shows all branches merged to the branch pointed by *HEAD* (these can be safely deleted)
`git branch --no-merged`	shows all unmerged branches to to the branch pointed by *HEAD* (not deletable)
`git branch --merged(--no-merged) my-branch`	shows all merged(unmerged) branches to branch *my-branch*

- Delete remote branch:
  `git push name-of-server --delete name-of-branch-to-delete`
- Show merge conflicts:
  `git mergetool`
- Show list of remote branches:
  `git ls-remote name-of-server`
  `git remote show name-of-server`
- More options on synchronising local database with remote server:

`git fetch --all`	each local branch gets updated
`git checkout -b name-of-branch name-of-server/name-of-branch`	creates a local branch copy of the remote one
`git checkout name-of-branch`	if *name-of-branch* exists on the server but not locally, it is created locally as a tracking branch of the one on the server
`git branch -u name-of-server/name-of-branch`	sets local current branch to track the specified remote branch

- Show local list of tracking branches:

`git branch -vv` *ahead/behind* tags may appear under the list entries: they are a reference to the position of the local branch with respect to the one on the server

- Rebase branch into another. Rebasing streamlines commits history, but is better done locally, never on server unless authorised [2 steps]:

  1. `git checkout branch-to-rebase` — position *HEAD* to the branch to be rebased
  2. `git rebase master` — rebase it into the *master* branch, for instance. Then it is easy to do a fast-forward merge with `git checkout master` followed by `git merge branch-to-rebase`

  alternatively

  `git rebase master branch-to-rebase` = `git checkout branch-to-rebase` + `git rebase master`]

3 Text editing from cmd/vim. Basic commands

- Create a text file and write something on it (will not append, just overwrite if file exists):
  `echo 'My text' > name-of-file.txt`
- Append a line to the body of an already existing text file:
  `echo ' added line' >> name-of-file.txt`
- Write text directly from *cmd* with vim:
  `vim text.txt`
- Save what was written on file with vim, exit vim and return to *cmd*:
  push *Esc* + `:wq` + push *Enter*