Git commands I've found useful - Git Cheatsheet

(note that things inside of '[]' are optional - except in the .gitconfig file)

`git add <file or dirname>` - add file to staged state
`git add -u` - add all currently tracked files under this directory to staged state
`git commit [-m "commit message"]` - commit all staged files into current branch (HEAD) (without the -m, your editor will pop up for you to enter a commit message)
`git status` - info about the current state of your git repo
`git whatchanged [-p][version1..version2]` - nice summary of what has changed [-p with filediffs]. Note that if you use the version limiting, version1 must be earlier in the repo than version2
`git show [<version>]` - show current commit [or commit at version]
`git show <version>:<filename>` - show file contents at version
`git diff [--stat]` - show diffs for all files you've changed but not staged (or the one line stats)
`git diff --staged` - show diffs for all files staged
`git diff <version1>..<version 2>` - diff of all files that changed between 2 versions
`git diff <version1>..<version2> -- filename` - diff between 2 versions of a specific file
`git diff --name-only <version1>..<version2> -` list of all filenames of files that changed between 2 versions
`git lg` **or** `qlol` or `git lol` or `git lola` or `git lolat` - see section below marked "LOLA"
`git branch [-d] <branchname>` - create [delete] a branch
`git branch [--merged|--no-merged]` - list all branches [that are merged | that aren't merged]
`git checkout <branchname>` - move the HEAD to the named branch, and reverts working files to it
`git branch -f <branchname> <version> -` move a branch to a different version (not checked out)
`git checkout -B <currbranch> <version>` - move branch and check out

`git merge <branchname>` - merge branchname into HEAD (current branch) and commit
`git merge --no-commit <branchname> -` merge branchname into HEAD, don't commit
If you tried a merge which resulted in complex conflicts and want to start over, you can recover with:
`git merge --abort`.
`git mergetool` - if there was a merge conflict, start the merge tool (has to have been set up)
`git checkout [--theirs|ours] -- <file>` - after a merge conflict, resolve by taking the file from the current branch [ours] or the branch you're merging from [theirs] (especially useful for a binary file)

`git grep <regexp> [-n] [<version>]` - grep for <regexp> in HEAD [<version>] [-n print line#]
`git grep <regexp> $(git rev-list --all)` - grep for <regexp> in all branches

`git tag [-l 'tagname'] [-n[#]]` - list tags in repo [of a name w/wildcard] [print n lines annotated]
`git tag -a <tagname> -m 'message'` - add annotated tag, give reason what tag is for
`git log [-p] [<filename>|<dirname>]` - any commits that modified file <filename> or anything under directory <dirname> [-p with file diffs]
`git log [-p][-2][...]` - list commits in repo.
`-p`: w/diffs, `-2`: last 2 entries, `--stats`: w/stats, `--shortstat`: mods from stats,
`--graph`: ascii history graph,
`--pretty=oneline|short|full|fuller|format:` various output formatting,
`--since=2.hours`, (lots more options)
`--raw`: list of filenames that changed in the commit

Git commands I've found useful - Git Cheatsheet

(note that things inside of '[]' are optional - except in the .gitconfig file)

`git log --follow <filename>` - list the history of a file, following renames/moves

`git rm <filename>` - stage a file to be removed
`git rm --cached <filename>` - remove a file from the staged state (& mark remove from repo)
`git mv <file-from> <file-to>` - move a file
`git checkout -- <filename>` - wipeout the modified file (unstaged)
`git checkout -b <branchname>` - create branch and move the HEAD to it (reverts working files too)
`git init` - create new git (without cloning)

From: https://git-scm.com/docs/git#_reset_restore_and_revert

Reset, restore and revert

There are three commands with similar names: `git reset`, `git restore` and `git revert`.

- git-revert[1] is about making a new commit that reverts the changes made by other commits.
- git-restore[1] is about restoring files in the working tree from either the index or another commit. This command does not update your branch. The command can also be used to restore files in the index from another commit.
- git-reset[1] is about updating your branch, moving the tip in order to add or remove commits from the branch. This operation changes the commit history.

  `git reset` can also be used to restore the index, overlapping with `git restore`.

Examples:
If you accidentally deleted file main.c, you restore it from the git db with: `git restore main.c`
If you want to undo the files changed in a commit, without deleting commits: `git revert <version>`

Accessing a remote repository:
Note that a git repository is a full access specifier (URL) to a repository. This is not the remote shortname, which is a shortcut stored by your local git repository hiding the URL for the remote server.
Remote commands:
`git clone <git repository>` - clone a remote git repository
`git ls-remote <git repository>` - look at branches, tags, and HEAD in a remote repository
`git fetch [remote]` - fetch data from the remote repo and update all of your tracked branches
`git pull [remote] [refspec]` - fetch and merge (from the remote repo it into your code [refspec can look like rbranch:lbranch]
`git push <remote repository> <lbranch>[:rbranch] [<tagname>|--tags]` - push local lbranch to the remote (useful for pushing to public repo), [and update the remote rbranch with it]

(the rest of these commands do not actually access the remote repo, but work on local refs to the remote repo)

Git commands I've found useful - Git Cheatsheet

(note that things inside of '[]' are optional - except in the .gitconfig file)

`git remote [-v]` - shows the remote shortnames you have configured
`git remote show <remote shortname>` - info about a remote repo, and your relationship to it
`git remote add <remote shortname> <git repository>` - add remote repo
`git remote rename <remote shortname1> <remote shortname2>` - rename your repo
`git remote rm <remote shortname>` - remove reference to a repo (entirely)

To figure out which commit (version) introduced a bug:
`git blame <filename>[version]` - for every line of the file, what was the last commit, who changed it [starting at version]
`git bisect start;git bisect good <version>;git bisect bad <version>`
git then moves you to a version that you test and report results with `git bisect [bad|good]` multiple times until git outputs the commit id of the bad commit. Use `git-show` for info on who did it, and then `git bisect reset` to go back to head. (there are additional bisect commands too, visualize, skip, etc)
note that there's a nice gui for this if you run `git gui blame`

after first installing git, be sure to set your user info
`git config --global user.name "John Doe"`
`git config --global user.email johndoe@example.com`
`git config --global core.editor emacs`

if you're in the middle of working on a branch, and you need to switch (change HEAD with the checkout command), but you're not ready to commit, you can stash away your work:
`git stash` - and you can see it with: `git stash list`
and to get it back `stash apply [stash@{2}]` - return to the most recent stash [or earlier one]
`git stash drop` - remove the stash from the list (apply doesn't do that)
`git stash pop` - apply the stash and remove it from the list

BE WARNED, if you do a get stash **drop** instead of a **pop**, you will lose your file changes! To recover, if you are very lucky, you can do this trick I found on stackoverflow:

If you didn't close the terminal, just look at the output from `git stash pop` and you'll have the object ID of the dropped stash. It normally looks like this:

```
$ git stash pop [...] Dropped refs/stash@{0} (2ca03e22256be97f9e40f08e6d6773c7d41dbfd1)
```

(Note that `git stash drop` also produces the same line.)

To get that stash back, just run `git branch tmp 2cae03e`, and you'll get it as a branch. To convert this to a stash, run:

```
git stash apply tmp
```

```
 git stash
```

Git commands I've found useful - Git Cheatsheet

(note that things inside of '[]' are optional - except in the .gitconfig file)

HEAD - where you are at HEAD^ parent HEAD~4 - g-g-grandparent...
untracked --- unmodified --- modified -- staged
.gitignore - file with names for git to ignore (esp. in status)

LOLA/LG
To see a text version of the branches, put this in the .gitconfig file:
[alias]

```
lg = log --graph --pretty=format:'%Cred%h%Creset -%C(auto)%d%Creset %Cgreen(%cd) %C(bold
blue)<%cn>%Creset %s ' --abbrev-commit --all
    lol = log --graph --decorate --pretty=oneline --abbrev-commit
    lola = log --graph --decorate --pretty=oneline --abbrev-commit --all
    lolat = log --graph --decorate --pretty=oneline --abbrev-commit --all --
    qlol = log HEAD~30..HEAD --decorate --pretty=oneline --abbrev-commit
```

[color]

```
branch = auto
diff = auto
interactive = auto
status = auto
```

Then type: `git qlol` or `git lol` or `git lola` or `git lolat <filename>`
Note that `lol` will only show your branch and ancestors of your branch, `lola` will show all branches in your repository, and lolat will who the branch history for <filename>. qlol is the same as lol, but will run much faster since it only shows the first 30 items from HEAD in the log.

THINGS TO DO BEFORE COMMITTING CHANGES:
`git diff --check` - check for whitespace errors

TO CREATE AN ARCHIVE OF YOUR GIT REPO
`(without creating a tar file)`
`git archive --format=tar <tag> <path> | tar -C <target dir> -xf -`
Note that this takes all the files in the git repo (starting at <path>) from the <tag> revision, and copies them into the <target dir>. You can use any version designator instead of just <tag>.
(if you want to actually create a tar file)

```
git archive master --prefix='project/' | gzip > `git describe
    master`.tar.gz
    creates the file v1.6.2-rc1-20-g8c5b85c.tar.gz from the annotated tag
```

Git commands I've found useful - Git Cheatsheet

(note that things inside of '[]' are optional - except in the .gitconfig file)

Random useful commands:

`git ls-files` - list all files in the index and under version control

`git checkout <branch> <path to file>` - checkout a file from another branch into workspace (still need to "git add" it)

`git cat-file -p <hashtag>` - Low level command to look at an object:

Commands to use very carefully - i.e. don't do it if you aren't sure

`git commit -a [-m "commit message"]` - stage all tracked modified files, then do a commit

`git commit --amend -m "new commit message"` - modify your previous commit (message and/or additional files you stage before this cmd)

`git revert <commit's hash#>` - does an undo of the commit (by adding a new commit)

`git push <remote shortname> <leave localbranch blank>:<rbranch> -` deletes remote branch <rbranch> from server:

`git rebase <branchname>` - warning - only use if you are positive you know what you are doing!! smashes together commits and branches together - makes your history look cleaner, but if you rebase any public/shared branch that's ever been pushed and then do a forced push, that's a nightmare in the making.

Learning Git:

https://git-scm.com/book/en/v2

https://www.atlassian.com/git/tutorials