Git Basic Commands

Get help about commands:
`git help [command]`
`git [command] --help`
`man git[-command]`

Setting username and email:
`git config [flags] user.name "Git User"`
`git config [flags] user.email gitisgreat@domain.com`
• You can use the flag `--global` if you want to set these for the user's whole system; otherwise, it's just set for the current repository.

Start a new repository:
`git init`
• It doesn't matter if you have files in the directory or not
• This creates a .git folder, which will store metadata about your files

Check the state of your repository:
`git status`
• If you don't want to see about an untracked file, you can make a file called .gitignore, which will not remind you about the untracked files you list in it.
• Contains lots of helpful messages

Stage a change for committing:
`git add [flag(s)] filename`
• * is a wildcard in the file name
• The `-N` argument allows you show your intent to add them, which allows you to later use `-a` for new files.
• You can stage parts of files! Use the flag `-p` to go into patch mode. Use the command `?` for information on commands and begin adding sections. Also see
`http://git-scm.com/book/en/Git-Tools-Interactive-Staging` for more info.

Unstage a file:
`git reset HEAD filename`
• This does not change the file in your working directory, but reset can with different arguments, so be careful using it

Commit file(s):
`git commit [flags]`
• The flag -a automatically stages modified/deleted files for this commit
• The flag -m allows you to write a commit message in-line:
```
git commit -m "Message here"
```
(generally, your commit messages should be more detailed than this)
• The flag `--amend` can be used if you goof up a commit. See

http://git-scm.com/book/en/Git-Basics-Undoing-Things as this can mess things up if you do it wrong, and you don't want to do it on things that have been published somewhere as it "rewrites history"

See the changes to a file:
`git diff [commit hash] [filename]`
• By default (with no arguments) shows the diff for all modified files
• This is the diff to the last staged (or committed if none are staged) version (this means that if you stage then run `git diff`, it won't show any difference)
• The flags `--staged` or `--cached` can be used to show difference between the currently staged files and previous commit(s)

Stage file removal and remove file from directory:
`git rm [flags] filename`
• directly deleting from the actual directory makes an unstaged change
• The flag `--cached` removes the staged snapshot of a file, but not the file itself

See commit history:
`git log [flags]`
• Commits are displayed in reverse chronological order
• The flag `-<number n>` shows the last n commits (eg. `git log -5`)
• The flag -p shows the diffs for the commits
• There are LOTS of log options, check out the book section on it!
http://git-scm.com/book/en/Git-Basics-Viewing-the-Commit-History
• The `--oneline` flag shortens output to one line per commit
• The `--graph` flag shows a neat graphical representation of the history

Update files in the current directory:
`git checkout [branch/commit] [--] [file_path]> [commit]`
• Used to change the current branch to another branch or change a file to a previous version
• This discards the changes in the current working directory when specifying a file, so be careful with it. You may wish to look up how to use it while gaining familiarity
• The `--` is used to specify a file is being checked out; just running `git checkout filename` will create a branch with that name.

Create a local copy of an existing repository:
`git clone repository_url`
• This repo's URL is `https://github.com/ccgarvey/Git-Presentation`

See a list of the remote repositories:
`git remote [flags]`
• The flag `-v` shows the URL for each remote
• The flag/phrase `-remote add shortname URL` adds a repository with an abbreviated name
• The flag/phrase `remote rename` renames a short name)

Get data from a remote repository that you don't already have:
`git fetch remote_name`
- This puts the data in the .git folder, so it doesn't modify your own files
- You need to merge later if there are conflicts (or if you want branches you don't have yet)

Fetch and merge a repository at the same time: `git pull remote_name`
- In older versions of git, you should commit before pulling (so do it always)

Share your work on the project:
`git push`
- If you clone, fetch, or pull and someone else pushes first, you need to pull before you can push your commits (it doesn't allow you to discard others' work)

Create, delete, or get information about branches:
`git branch [flags] [branch name]`
- With no arguments, git branch lists the current branches
- With just a branch name (ex. "experimental"), creates a branch (but does not switch the current branch)
- To switch between branches, use `checkout` (Note: you must commit changes before switching)
- The flag `-d` deletes a branch
- Branches must be pushed to share:
  `git push remote_name localbranch[:remotebranch]`

Merge disparate files or branches:
`git merge`
- If you get an auto-merge with conflicts, you get info in your files you must resolve, you can use `git mergetool` to merge if you would like
- You must commit your merge after completing it.
- Don't merge with uncommitted changes in your local directory!
- When you fetch and it gets remote branches, you must merge the branches in
- Rebasing is another way to combine divergent histories (see below)

Merge branch in a way that produces linear commit history (rebase):
`git rebase branch_to_rebase_on`
- *NEVER PUSH REBASED CODE TO A PUBLIC REPOSITORY.* (rewrites history)
  > "If you [do], people will hate you, and youll be scorned by friends and family."
  > –Git Pro, `http://git-scm.com/book/en/Git-Branching-Rebasing`
- This rebases your current branch onto the branch specified
- This finds the common ancestor of the two branches involved, finds the differences between the two, and applies them to the branch being rebased onto so that there are new commits ending with the files as they are in your current branch (if this is confusing, try the git pro book link provided)
- When the rebase is done, merge is used and automatically combines the branches:

```
git checkout branch_rebased_on
git merge
```

- This produces a linear history, instead of a branched-joined history

Git-related Resources

Git website:
`http://git-scm.com/`

Getting Git:
Linux (if not pre-installed):
`apt-get install git`
`yum install git-core`

Mac:
With MacPorts: `sudo port install git-core +svn +doc +bash_completion`
Using a GUI installer: `http://code.google.com/p/git-osx-installer`

Windows:
Git bash + GUI installer: `http://msysgit.github.com/`

Git Pro book:
`http://git-scm.com/book`

Try Git in-browser:
`http://try.github.io`

Websites for hosting git code:
- GitHub: `https://github.com/`
- BitBucket: `https://bitbucket.org/`