A Short Guide to Git

This is a quick, short guide to using Git. Consider this a pick of the top few important commands that you will end up relying on.

Creating a Repository

Creating a repository is simply a matter of jumping into the appropriate directory and asking git to initialise itself. You should do this for every project (little, or large). After a while you'll find you do it even for tiny scripts.

```
$ cd ~/your_project
$ git init
```

Adding files

Without adding, the file to Git, git will just ignore it. Here's an example of adding a README:

```
$ vim README.md
$ git add README.md
```

You'll need to do this for every file you want to manage with Git.

Notably, with GitHub you can format README's in various different markup languages (Markdown is shown here), it will then be formatted on the repository page.

Committing Changes

Committing is as simple as:

```
$ git commit
```

You can use the "a" flag to commit all changed (but tracked) files:

```
$ git commit -a
```

And, the "m" flag to add a simple message. Without the "m", it will open in your shell's standard editor.

```
$ git commit -m 'A commit message.'
```

Undoing Changes You're Yet to Commit

When you add files, you are adding them to the staging area.

Occasionally, you might stage something to be committed that you don't want to. You can reset a staged file like so:

```
$ git reset README.md
```

Similarly, you may wish to undo all of your changes and roll back to a previous commit:

```
$ git reset --hard HEAD
```

Or, rollback to a specific ref (you can find this using `git log`):

```
$ git reset cdbd65d
```

Using Remotes

Once you have committed something to your repository, it's time to push it to GitHub. The following is taken from the guide GitHub provides once you create a repository.

```
$ git remote add origin git@github.com:nickcharlton/test.git
$ git push origin master
```

These commands add a remote for you to push the code, and specify which branch to push to it. If you omit the "origin" alias, git will push to the last remote you used.

Git can use ssh, rsync, http and the file:// url scheme for communicating with remotes. GitHub (and almost everywhere else) uses ssh.

Branching

The great flexibility of git is through it's power to create, merge and delete branches.

Branches allow you to take your current code base, do something different to it and then merge it back into master. A common workflow is to branch for each new feature, or bug fix, or experiment and then merge back into master afterwards.

```
$ git branch
```

Lists all of your current branches.

```
$ git branch new_feature
```

Would create a new branch called new_feature with the contents of your current codebase. Then, issuing `git checkout new_feature` will switch you to the new branch.

```
$ git checkout new_feature
```

Or, you can use `git checkout -b new_feature` and it'll create the branch and switch to it for you.

Deleting a Branch

```
$ git branch -D experimental
```

Fetching & Pulling

`fetch` pulls down a remote branch.

```
$ git fetch experimental
```

It’s then up to you to merge the fetched branch into the rest of your working tree.

There is also a “pull” command which combines a “fetch” and a “merge” with the current branch you are on. It works well for syncing small changes that are unlikely to merge badly. For example:

```
$ git pull
```

Merging

Merging is combining two branches.

This is done by calling a 'merge' in the branch you want to merge with.

```
$ git merge new_feature
```

For example, whilst in the master branch, merging the experimental branch will pull in the code and make it live in harmony with your previous work. If none of the commits clash, they will be replayed on top of each other in what’s known as a “fast-forward merge”. This is what’ll happen most of the time.

Or, it won't and you'll end up with a merge conflict. At first they seem daunting, but with practice they are relatively simple to pick apart. The process is vaguely as follows:

1. See what clashed by checking the current repository status.
2. Edit the relevant files to pick the preferred section of code.
3. Commit all of the changes.

See: http://book.git-scm.com/3_basic_branching_and_merging.html

Syncing with a Remote Repository

If you have a repository being edited by others on a remote location, such as GitHub, to sync with your current branch, you can either run `git pull` or `git fetch` followed by `git merge`, the documentation suggests doing the latter for flexibility.

```
$ git fetch git@github.com:[user]/[repo].git
$ git merge origin/master
```

Where [user] and [repo] represent the repository URL, and in the second line, you are merging with your current branch (this is by default), then the branch for which you pulled down. With GitHub, this is `origin/`, then the name of the repository, so for the main repo, `master`.

Tagging

Tagging is used when a specific bit of code reaches a certain milestone. Such as a version number. It's especially useful for packaging up releases, and for keeping issues tacked to a specific version.

This tags the last commit, with the name "v1.0".

```
$ git tag v1.0
```

You can also tag specific commits (`git tag -a v0.9 a2fcec4`), which is fantastic when you forget to mark a version.

And, you can add a message to a tag - handy for release notes.

Pushing tags

If you're using a remote (i.e., GitHub), you'll likely want to push your current tags. You can also push a specific tag, this is shown on the second line. Replace `REMOTE` and `TAG` with whichever is relevant.

```
$ git push REMOTE TAG
$ git push REMOTE BRANCH --tags
```

Going Further

This really is just the surface of what Git can do. With use, you will find that Git is stupidly flexible. There's also a lot of stuff that has been written about Git, GitHub and version control. Some recommended places to read, and resources to check out are below.

- Pro Git by Scott Chacon: http://progit.org/book/
- Git Reference: http://gitref.org/
- Version Control for Designers: http://hoth.entp.com/output/git_for_designers.html
- The "Concept of Git": http://stackoverflow.com/questions/9204168/understanding-the-core-concept-of-git/9561014#9561014

If this guide has left you hungry for more, I suggest you read up on the following:

1. git rebase: Allowing you to rewrite history, by merging and picking specific commits.
2. git submodules: Allowing you to include another repository inside your current one. This is especially useful for including libraries.
3. And, finally: Configure git. Enabling colours is a good start.

Version History

Date	Changes
09/03/2012	Clarifies most of the wording. Regroups to follow along with the associated presentation. Adds links to resources, a version history and licensing.
21/02/2011	Initial Version. Covers making repositories, branching, merging, remotes & tags.

Author & License

Come across this from elsewhere? You can find the original here: http://nickcharlton.net/posts/git-workshop