Josh's Introduction to Git

In modern software development and documentation, the version control system *git* is most often used for source control. This system is based on two concepts: repositories and branches. A *repository*, also known as a *repo*, is a collection of files. A *branch* is a version of those files that can be edited independently of other versions of those files. A repo holds one or more branches. Often, though, it contains many branches, allowing different users to edit, delete, and add to different branches at the same time without overwriting other users' work.

Since each developer (and writer) contains a copy of the repo on their local machine (a *local repo*), they edit the *local branches* in these repos without requiring network access. When a developer is ready to share their work with other users, they upload their local branch to a repo accessible by the other users, the *remote repo*, which contains *upstream branches*. Developers frequently download and upload work from the remote repo. In this way, many different developers can work independently while sharing their work when it is complete.

Commits: the building blocks of git

As mentioned previously, repos contain branches, which are different versions of the files in the repo. Git stores the information about branches in *commits*. When a user alters the files in a branch and then wants to save those changes in the git repo, the user *commits* those changes. This results in a commit, which contains the list of files changed, and the exact changes to those files since they were last committed. Commits do not contain the files in the repo. Instead, each commit only contains the changes to the branch since the last time files in it were committed. The changes from one commit to another is referred to as the *diff* between those two commits, and git can compute and show you the diff between any two commits in a repository -- not just a commit that occurred directly after another commit. By viewing diffs, you can see how your project has evolved from any point to any other point in time.

Each commit is based on one or more previous commits. When a commit is based on a single commit, the latest commit updates a branch that contains the previous commit. When the commit is based on multiple commits, this is the result of two branches merging.

While we often use branches, because they are usually easier to refer to than referring to commits, each branch is fundamentally a collection of ordered commits, since each commit is based off of one or more ancester commits. Git computes the diff from the very first commit in the branch to the current state of the branch -- the *tip* of the branch -- when you view or edit a branch. This diff includes the total changes in the branch to all of the files in the repo, including additions and deletions within files, and additions and deletions of entire files, too.

Basic operations

While git provides many commands and options, you will use the following operations repeatedly:

- `git push`: Uploads changes from a local branch to an upstream branch.
- `git fetch`: Downloads information from a remote repo about the current state of its branches.
- `git merge`: Inserts material from one branch into another branch.
- `git pull`: Performs `git fetch` and then performs a `git merge` from the upstream branch into your local branch.

The following sections briefly describe each of these operations. For more detailed options for each of these operations, please see the official git documentation online. Also, note that in the following sections, we assume that you have set up your local branches to track upstream branches. For more information on setting up tracking branches, see this page.

`git push`

`git push` uploads changes from your local branch to the upstream branch so that they are identical. Since the upstream branch is in the remote repo (which other users have access to), using `git push` shares your work with everyone who has access to the remote repo. Note that this operation requires your local branch to contain all of the commits that the remote branch contains *before* running `git push`, and will fail if this condition is not met. If the remote branch contains commits that your local branch does not yet contain, you have two options: (a) first run `git pull`, or (b) first run `git fetch` followed by `git merge`. After doing (a) or (b), you can run `git push`.

Syntax: To upload changes from your local branch `hotfix` to the upstream branch `hotfix` in the remote repo named `origin`, run

```
git push origin hotfix
```

`git fetch`

`git fetch` downloads information about the current state of the remote repository, including all of the new commits and branches it contains. This operation is always safe to run, however, because it does not *change* your local branch. To update your local branch with changes in the upstream branch, first run `git fetch` and then run `git merge`.

Syntax: To download information from the remote repository named `origin`, run

```
git fetch origin
```

`git merge`

`git merge` compares two branches and updates one of them with changes that the other one contains. This operation can be used to update one branch with information from any other branch, including upstream branches and local branches.

Syntax: To update the local branch `master` with changes in the upstream branch `hotfix`, first switch to `master` by running

```
git switch master
```

and then execute the merge by running

```
git merge hotfix
```

`git pull`

Frequently you will want to perform a a `git fetch` immediately followed by a `git merge`, which is exactly what `git pull` does. The advange of first performing a `git fetch` and *then* performing a `git merge` is that you can run `git status` between those two operations to determine whether or not running `git merge` will result in a merge conflict. (A merge conflict occurs during `git merge` when git cannot automatically integrate the changes introduced by the merge because the two branches contain commits that conflict with one another. When a merge conflict occurs, you must instruct git how to resolve the conflict.)

Syntax: To update the local branch `master` with information from the remote branch `master` in the remote repository named `origin`, first switch to `master` by running

```
git switch master
```

and then execute the pull by running

```
git pull origin master
```

Further information

This introduction to git covered several basic git commands, but there are many others, as well as many options for each command. By leveraging additional commands and options, git becomes not just a version control system but a set of tools that assists in your work. For instance, git allows you to re-order your work so that it is more understandable to you and your peers. The system also allows you to do work that you need temporarily -- for instance, importing test files -- and later undo that work by undoing a commit. To fully understand git, it's best if you first take the time to understand the three trees. Once you have a solid understanding of that topic, you may be interested in learning about these additional git commands:

- `git diff`
- `git switch`
- `git stash`
- `git restore`
- `git revert`
- `git reset`
- `git rebase`

You can find information on all of these commands as well as all other git commands, at https://git-scm.com/doc.