## Git Tag

### Lightweight Tags

$ git tag stable-1 1b2e1d63ff

This creates a "lightweight" tag, basically a branch that never moves

### Tag Objects

If one of **-a** the command creates a tag object, and requires the tag message.

When this happens, a new object is added to the Git object database and the tag ref points to that tag object, rather than the commit itself.

## Ignoring files

You can tell git to ignore certain files by creating a file called .gitignore in the top level of your working directory, with contents such as:

## Interactive Rebasing

If you have a number of commits that you would like to somehow modify during the rebase, you can invoke interactive mode by passing a '-i' or '--interactive' to the 'git rebase' command.

$ git rebase -i origin/master

If 'pick' is specified, it will simply try to apply the patch and save the commit with the same message as before.

If 'squash' is specified, it will combine that commit with the previous one to create a new commit.

## Interactive Adding

$>git add -i

If you type '5' or 'p' in the menu, git will show you your diff patch by patch (or hunk by hunk) and ask if you want to stage each one. That way you can actually stage for a commit a part of a file edit. If you've edited a file and want to only commit part of it and not an unfinished part, or commit documentation or whitespace changes seperate from substantive changes, you can use 'git add -i' to do so relatively easily.

## Stashing

$ git stash

$>git stash list

## Git Treeishes

There are a number of ways to refer to a particular commit or tree other than spelling out the entire 40-character sha. In

## Git, these are referred to as a 'treeish'.

^ will give you the Nth parent of a particular commit. This format is only useful on merge commits - commit objects that have more than one direct parent.

master^2

The tilde spec will give you the Nth grandparent of a commit object. For example,

master~2

Finding files with words or phrases in Git is really easy with the linkgit:git-grep command.

$ git grep adel

If I wanted to see the line number of each match as well, I can add the '-n' option:

$ git grep –n adel

If we're only interested in the filename, we can pass the '--name-only' option:

$>git grep --name-only adel

We could also see how many line matches we have in each file with the '-c' option:

$>git grep -c adel

Now, if I wanted to see where that was used in a specific version of git, I could add the tag reference to the end, like this:

$ git grep xmmap v1.5.0

We can also combine search terms in grep.

$ git grep -e 'adel' --and -e saki

We can also search for lines that have one term and either of two other terms, for example, if we wanted to see where we defined constants that had either PATH or MAX in the name:

$ git grep -e '#define' --and \( -e PATH -e MAX \)