Git in Ten Minutes

Spencer Tipping

August 31, 2010

Contents

1 Commits

As soon as you hand your files over to Git, Git keeps copies. You can nuke your working tree, edit it arbitrarily, and unless you tell Git to absorb those changes using `git commit`, Git really doesn't care. This is probably the most misunderstood thing about Git, at least at first. But once you wrap your mind around it things start to make sense. So, for example, here's a Git session along with how it sees the world.

1.1 How Git sees your files

```
$ git init
```

Git: Ok, I've created a directory called `.git` where I'll store mysteriously-compressed versions of all of the files you give me. You can then bork anything else, and I'll be able to reconstruct any file you've given me from the stuff in `.git`.

```
$ echo 'test' > foo
```

Git: You've created a file called `foo` that I know nothing about and I won't touch with a ten-foot pole.

```
$ git add foo
```

Git: Aha, so you *do* want me to manage `foo`. Well certainly. I've taken a snapshot of `foo` and placed it into the *index*, which is a place where you can put snapshots of things as you're getting ready to commit them. If you edit `foo` from here, I won't see those changes unless you do another `git add`.

```
$ git commit
```

Git: Ok, I'm going to make a commit out of everything in the index. In this case, you've handed me a new file, so I'll commit the fact that you created `foo` and that its contents were `test` when it was added to the index. (For all I know, it might be something else now; but you wanted me to commit the snapshots in the index, so that's all I did.)

```
$ echo 'test2' > foo
```

Git: You've made changes to `foo`. I don't own any of those changes, though, since you haven't made a new snapshot. As far as I'm concerned, `foo` is supposed to contain `test`. If you want me to create another snapshot and absorb those changes, you'll have to use `git add`.[1]

```
$ git diff foo
```

Git: You're asking me how `foo` is different from the way I was expecting it. Equivalently, how `foo` will change if you use `git add`. I'll print a diff to show what has changed between your copy of `foo` and the last current `foo` that you've given me.

```
$ git add foo
```

Git: Ok, I've taken `foo`'s current contents and put them in the index. If you run `git commit`, I'll create a commit out of that change.

1.2 Common shortcuts

Git provides some shortcuts to access the most commonly-used functionality:

```
$ git commit <filename> [<filename> ... <filename>]
```

Git: I'll take a new snapshot of each of those files (provided that I own it; you must have `git added` it at least once for me to be willing to touch it) and create a commit from those snapshots. After this, any other snapshots in the index will still be uncommitted, but all changes that you've made to the files you just specified will be committed.

```
$ git commit -a
```

Git: I'll commit any changes to every single file you've ever `git added`, along with anything you've put into the index.

[1] There is an exception, really a shorthand: `git commit -a`, which I'll go over later.

2 Branches

Before I go into branches there's something you should know about commits. Each commit represents changes to a file (or many files), and changes only make sense if you know what the file looks like. So if you make two commits, first commit `A` and then commit `B`, then commit `B` knows that it depends on commit `A`.[2] You can see, then, that the most recent commit has a chain all the way back to the beginning of your repository.

But all of this raises a question: How does Git keep track of the most recent commit? The answer is that it uses a label. When you create a repository, that label is `master`; and Git updates `master` each time you make a new commit. It's appropriate to think of `master` as a symlink that Git keeps pointing at the most recent commit.

[2] In practice, the commit names `A` and `B` are very long hexadecimal numbers computed from the diffs represented by the commits, and Git can look up the data in the commits based on these numbers.