A Simple Example

coursera.org/learn/git-distributed-development/supplement/vRgrR/a-simple-example

Let's get a feel for how git works and how easy it easy to use. For now, we will just make our own local project.

First, we create a working directory and then initialize git to work with it:

```
3

$ git init
```

Initializing the project creates a **.git** directory which will contain all the version control information; the main directories included in the project remain untouched. The initial contents of this directory look like:

```
1

2

3

4

5

6

7

8

9

10

11

12

$ ls -l .git

total 40

drwxrwxr-x 7 coop coop 4096 Dec 30 13:59 ./

drwxrwxr-x 3 coop coop 4096 Dec 30 13:59 ../
```

```
drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 branches/
-rw-rw-r-- 1 coop coop   92 Dec 30 13:59 config
-rw-rw-r-- 1 coop coop   58 Dec 30 13:59 description
-rw-rw-r-- 1 coop coop   23 Dec 30 13:59 HEAD
drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 hooks/
drwxrwxr-x 2 coop coop 4096 Dec 30 13:59 info/
drwxrwxr-x 4 coop coop 4096 Dec 30 13:59 objects/
drwxrwxr-x 4 coop coop 4096 Dec 30 13:59 refs/
```

Later, we will describe the contents of this directory and its subdirectories; for the most part, they start out empty.

Next, we create a file and add it to the project:

```
2
$ git add somejunkfile
```

We can see the current status of our project with:

```
7
8
9
10
11
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file: somejunkfile
#
```

Notice it is telling us that our file is staged, but not yet committed.

Master vs Main

Historically, a new repository was always created with an initial branch called **master**. However, many projects have deprecated this name and replaced it with **main**.

Reasons for this change and other changes of terminology, such as deprecation of the use of the words blacklist and whitelist, are explained in a document created by the **Inclusive Naming Initiative**.

Technically, this main branch could have any name, as there is nothing structurally special about this authoritative branch. However, git has to have a default name for new repositories, and some web hosts have requirements that have nothing to do with git itself.

GitHub has strongly recommended this naming convention change and has given instructions at https://github.com/github/renaming about how to create new repositories as well as rename old ones to have main as the main branch.

If you are creating a new repository, the easiest way to do this from the outset is to type something like:

```
1

2

$ git init

$ git checkout -b main
```

For existing repositories, you can rename both the local branch and the remote branch on the server with:

```
1
2
3
4
5
6
7
8
9
10
$ git checkout master
# Change the local name
$ git branch -m master main

# Change the remote name
$ git push -u origin main
$ git symbolic-ref refs/remotes/origin/HEAD refs/remotes/origin/main

# Confirm the names!
$ git branch -a
```

Depending on the setup of the remote server (such as GitHub) this may or may not work. An easier approach is to not delete the master branch but just copy it to main and work from there from now on, as in:

```
4
$ git push -u origin main
```

and then just ignore master from now on.

From now on we will assume that the name of the branch that plays this role is *main* and that any necessary changes have been made.

Let's tell git who is responsible for this repository:

```
2

$ git config user.email "b_genius@linux.com"
```

This must be done for each new project, unless you have it predefined in a global configuration file.

Now, let's modify the file, and then see the history of differences:

```
1

2

3

4

5

6

7

8

9

$ echo another line >> somejunkfile

$ git diff

diff --git a/somejunkfile b/somejunkfile

index 9638122..6023331 100644

--- a/somejunkfile

+++ b/somejunkfile

@@ -1 +1,2 @@
```

```
some junk

+another line
```

To actually commit the changes to the repository, we do:

```
6

 create mode 100644 somejunkfile
```

If you do not specify an identifying message to accompany the commit with the **-m** option, you will jump into an editor to put some content in. You must do this or the commit will be rejected. The editor chosen will be what is set in your **EDITOR** environment variable, which can be superseded with setting **GIT_EDITOR**.

You can see your history with:

```
7

3

4

5

6

1

2

    My initial commit

commit eafad66304ebbcd6acfe69843d246de3d8f6b9cc

Author: A Genius <a_genius@linux.com>

Date:   Wed Dec 30 11:07:19 2009 -0600


$ git log
```

and you can see the information you got in there. You will note the long hexadecimal string, which is the commit number; it is a 160-bit, 40-digit unique identifier which we will discuss later. git cares about these beasts, not file names.

You are now free to modify the already exiting file and add new files with **git add**. But they are staged until you do another **git commit**.

Now, that was not so bad. But we have only scratched the surface.