Local Git Repo - #1 The Basics

#git-workshop

Getting started

```
$ mkdir demo_folder
$ cd demo_folder
```

Git commands all look something like `git [command name] [possible options]`. So keep an eye out for that!

The first thing we do is setup a new repository. If you haven't already, please also set your name and email at this point. The git config should be a one time thing – if you've done it before, no need to do it again for new repositories.

```
$ git init
$ git config --global user.email "you@example.com"
$ git config --global user.name "Your Name"
```

Now try out the following in the repo.

```
$ git status
```

This will show us that there's nothing in this repo, as expected. If we do

```
$ ls -l
```

we can see that a `.git` folder has been created. Don't delete this folder! You should almost never need to open it up for normal Git operation.

Creating files

Let's create some files. This will create a new file named `README.md` in the current folder (don't worry if you don't understand these commands).

```
$ echo "Hello World" > README.md
```

Let's check what's going on. This shows that we have new changes in the current folder.

```
$ git status
```

Here's where the magic happens. We're going to *add* our new file, and then create a *commit*. This saves it for later.

```
$ git add README.md
$ git commit -m "Add README.md"
```

These commands will show us that our changes have been saved to the repository history.

```
$ git log
$ git status
```

Notice that we've been running `git status` a lot. In general, this is a good practice since status is a "safe" git command (as in, you won't break anything in your repo by doing `git status`). The same goes for `git log`.

It might not seemed like we accomplished much. But this is really all you need to add files to Git. We can add multiple files before committing too:

```
$ touch pangram1.txt
$ touch pangram2.txt
$ git add *.txt # Or git add pangram1.txt; git add pangram2.txt
$ git commit # This will open up an editor for the commit message. You can also use -m as before.
```

Changing files

Lastly, Git isn't just about adding new files, it also tracks file changes. Let's add some text to

one of our new files:

```
$ echo "The quick brown fox jumps over the lazy dog" > pangram1.txt
$ echo "Watch 'Jeopardy!', Alex Trebek's fun TV quiz game" > pangram2.txt
git status
```

Git will now show us we've made changes to both `pangram1.txt` and `pangram2.txt` that haven't been saved. You can verify that the changes are what you expect by using:

```
$ git diff
```

We can now commit the changes for the first file:

```
$ git add pangram1.txt
$ git commit # This will open up an editor. You can also use -m as before
```

Check the status and log again. It'll show that we saved the changes for `pangram1.txt`, but `pangram2.txt` hasn't been committed.

```
$ git status
$ git log -p
```

Finally, let's go ahead and actually commit the second file too. Here's a shortcut for adding all the changes in the current directory:

```
$ git add .
$ git commit
```