Exercise 5.1: Version Control with git

Your system may already have **git** installed. Doing `which git` should show you if it is already present. If not, while you may obtain the source and compile and install it, it is usually easier to install the appropriate pre-compiled binary packages; your instructor can help you identify the needed packages if they are not already installed or cannot be installed with one of the following commands:

```
$ sudo yum install git*
$ sudo zypper install git*
$ sudo apt install git*
```

according to your particular distribution.

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.

1. First we create a working directory and then initialize **git** to work with it:

   ```
   $ mkdir git-test
   $ cd git-test
   $ git init
   ```

2. 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:

   ```
   $ 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.

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

   ```
   $ echo some junk > somejunkfile
   $ git add somejunkfile
   ```

4. We can see the current status of our project with:

   ```
   $ git status
   ```

   ```
   # On branch master
   #
   # Initial commit
   #
   # 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**.

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


```
$ git config user.name "Another Genius"
$ 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.

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

```
$ 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
```

7. To actually commit the changes to the repository we do:

```
$  git commit -m "My initial commit"
```

```
Created initial commit eafad66: My initial commit
 1 files changed, 1 insertions(+), 0 deletions(-)
 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`.

8. You can see your history with:

```
$ git log
```

```
commit eafad66304ebbcd6acfe69843d246de3d8f6b9cc
Author: A Genius <a_genius@linux.com>
Date:   Wed Dec 30 11:07:19 2009 -0600

    My initial commit
```

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

9. 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`

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

