Git exercises

Configuring Git

We do this by setting a couple of options in a file found in your home directory

```
git config --global user.name "Firstname Lastname"
git config --global user.email username@company.extension
```

Your name and email address is included in every change that you make, so it's easy to keep track of who did what

Also, unless you are a vimwizard, I would recommend changing your default editor to nano

```
git config --global core.editor nano
```

If you have a different favorite editor, you can type in the appropriate command from the table below:

Editor	Configuration command
Atom	`$ git config --global core.editor "atom --wait"`
nano	`$ git config --global core.editor "nano -w"`
Text Wrangler	`$ git config --global core.editor "edit -w"`
Sublime Text (Mac)	`$ git config --global core.editor "subl -n -w"`
Sublime Text (Win, 32-bit install)	`$ git config --global core.editor
"'c:/program files (x86)/sublime text 3/sublime_text.exe' -w"`	
Sublime Text (Win, 64-bit install)	`$ git config --global core.editor

"'c:/program files/sublime text 3/sublime_text.exe' -w"`	
Notepad++ (Win)	`$ git config --global core.editor "'c:/program files
(x86)/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"`	
Kate (Linux)	`$ git config --global core.editor "kate"`
Gedit (Linux)	`$ git config --global core.editor "gedit -s -w"`
emacs	`$ git config --global core.editor "emacs"`
vim	`$ git config --global core.editor "vim"`

Make sure everything was entered correctly by typing `git config --list`

```
user.name=Dawn Childress
user.email=kirschbombe@gmail.com
core.editor=nano
```

A new Git repository

First, verify that you are in your Home directory using the `pwd` command (print working directory)

```
$ pwd
/Users/kirschbombe
```

1. To verify that we are not currently in a Git directory, we can use `git status`

```
$ git status
fatal: Not a git repository (or any of the parent directories): .git
```

2. Let's create a new project folder named `oceans` and initialize it as a Git repository

```
$ git init oceans
Initialized empty Git repository in /Users/kirschbombe/oceans/.git/
```

Using `git init oceans` creates the new folder "oceans" and initializes it as a git repository at the same time. We could also navigate into any existing directory and use `git init` to turn it into a git repo.

3. Now let's check that our new directory is there and move into it using the `ls` and `cd` commands

```
$ ls
oceans
$ cd oceans
$ pwd
/Users/kirschbombe/oceans
```

4. Right now there is nothing (except our .git file) in our directory, so let's create a new file named `mammals.txt` using the `touch` command and edit the file using `nano`

```
$ touch mammals.txt
$ ls
mammals.txt
$ nano mammals.txt
```

This will open the file in nano. Let's add some content to our file:

```
dolphins
whales
seals
```

Use `CTRL + O` to write, and `CTRL + X` to exit.

Now let's see what git is up to with `git status`

```
$ git status
On branch master
```

```
No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    mammals.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Here, git is telling you that there are local files that you haven't told git to look at. Let's tell git to add our changes to the staging area using `git add`

```
$ git add mammals.txt
```

Now, when we run `git status`, git tells us that we've told it to keep track of a new file

```
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   mammals.txt
```

If this was a mistake, we could correct it with `git rm`

5. Now we're ready to `commit` this file. Committing changes means making a permanent record of the current state of your repository.

```
$ git commit -m "add mammals.txt"
[master (root-commit) 579efa2] add mammals.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 mammals.txt
```

We use the `-m` flag followed by a **commit message** when we make a commit. Every commit needs a message to accompany it. It should be as brief as possible while still describing what changes you made.

> Making a whole bunch of changes and committing them all at once is *BAD*. If you make many commits for many small changes, and one of those changes breaks your code,

> you can *selectively undo* that change and fix the bug. If you make only one commit, you'll have to re-do everything from scratch :(

It takes a while to get a hang of this, and it helps to read other people's commits to know what to say

6. Take a few minutes to make some changes to your mammals.txt file and add a new file to your folder. Then add the changes to the staging area and commit your changes following the steps you just took in 4-5.

7. To view your commit history, use the command `git log`

```
$ git log
commit 579efa22f5e5f2c76dfcf04279b484408b52b22a (HEAD -> master)
Author: Dawn Childress <kirschbombe@gmail.com>
Date:   Sun Jan 28 20:36:38 2018 -0800

    add mammals.txt
```