Git basics lab

In this lab we are going to carry out some basic git operations.

You can complete this work in two ways:

- Use the Coursera Visual Studio Code environment in the browser
- Use your own installation of git.

Please see previous reading which contains links and instructions about installing git on your own machine.

Gitting started

Now we are going to assume you have git installed, or that you are in the Coursera Visual Studio Code environment.

Start a terminal.

- Mac: run the Terminal app
- Windows: start a Power Shell or WSL terminal
- Linux: run your preferred terminal
- Coursera: select Terminal from the View menu

In the Terminal, type the following command:

```
git status
```

If you are *not* in a folder that is under git version control, you should see this message:

```
fatal: not a git repository (or any of the parent directories): .git
```

If you are in a folder under git version control, you should see a message which starts something like this:

```
On branch master
```

If you see a message warning that the git command is not found, you have not installed git properly and you need to go back to the installation instructions in a previous reading.

Create a new git repository

Ok we are assuming you are in your terminal, in a folder that is not under git control and you have git working.

Create a new folder for your new repository (repo), go into it and initialise the repo:

```
mkdir my-project
cd my-project
git init
```

Check the status:

```
git status
```

Ask git to track one of your files

Create and save a file called README.md in the 'my-project' folder. You can do this with your regular text editor, or if you are working in our labs environment, use Visual Studio Code's file tab to do it. Put some text into the README.md file.

Now add the file to the repository in the terminal:

```
git add README.md
```

Check status:

```
git status
```

You should see something like this:

```
On branch master

No commits yet

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

Now git is tracking your README.md file.

Un-stage a file

Can you figure out how to stop git from tracking your README.md file? Clue - read the status message.

Commit changes

If you un-staged README.md, re-stage it with the git add command.

Create a commit, which is a snapshot of the state of the repository that you can return to later.

```
git commit -a -m "first commit"
```

- -a : commit all tracked files
- -m : specify a message for the commit (otherwise, it'll fire up your text editor so you can write a message there)

The first time you run this command, git will prompt you to provide some identification details which will be added to your git profile on your local machine. These details will then be attached to your commits. On your first ever commit on a machine, you should see something like this:

```
*** Please tell me who you are.

Run

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

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'root@a00e6836ca5f.(none)')
```

As you can see, git is telling you which commands you need to run to 'tell it who you are'. Go ahead and run these commands, putting in appropriate details:

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

What I tend to do with the email is to put the name of the machine I am working on, e.g. matthew@yogurt. I do that so that I can see which machine I sent the commit from in the commit log.

Ok so once you've run those commands to identify yourself, re-run the commit command.

```
git commit -a -m "first commit"
```

You should see something like this:

```
[master (root-commit) 55a047f] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

Check status

```
git status
```

You should see something like this:

```
On branch master
nothing to commit, working tree clean
```

Check the commit log:

```
git log
```

You should see your commit. Try this for a more compact view:

```
git log --pretty=reference
```

Branching

Now let's try and create some branches. We will create a branch, add a new file to it, then merge it back into the master branch. This is a standard, everyday workflow in git.

Work on a branch

Creata and switch to a new branch:

```
git checkout -b first-branch
```

(the branch is called first-branch)

List branches:

```
git branch -l
```

You should see something like this:

```
* first-branch
  master
```

The * tells you you are on first-branch

Create a new file called index.html in your folder. Put some content in that file. Add it to the repository:

```
git add index.html
```

Check status:

```
git status:

On branch first-branch
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   index.html
```

Commit your changes to the branch:

```
git commit -a -m "added index file"
```

Merge back to the master branch

Imagine we have now completed our work on the branch and we want to merge it back into the main branch. Go back to the master branch:

```
git checkout master
```

Notice there is no '-b' option here. Why is that?

Check the contents of the directory using your file explorer or in the terminal with ls -l or dir. Verify that the index.html file is not there. You can see that git is actually making changes to the files on the filesystem.

Now let's bring index.html in from the other branch:

```
git merge first-branch
```

You should see something like this:

```
Updating 55a047f..23ecd0d
Fast-forward
 index.html | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 index.html
```

Verify using your file explorer that the index.html has appeared in the folder. Use git log to verify that the two commits are now visible on the master branch.

All done. If you don't want to leave that branch hanging around, delete it:

```
 git branch -d first-branch
```

Deal with a conflict

The final thing we are going to do is to deal with a merge conflict. This is when the auto-merge fails and we need to manually fix it. I am going to leave out some of the commands so that you have to figure them out for yourself.

Create and switch to a new branch:

```
git checkout -b second-branch
```

Make an edit to line 1 of the README.md file. Commit your changes.

Switch back to the master branch and make an edit to line 1 of the README.md file. Note that you master branch version of README.md is different from the second-branch version as you have not merged yet.

Now attempt to merge the second-branch into the master branch. You should see a message a bit like this:

```
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

This is because git does not know which line 1 of README.md to accept - the one in master or the one on second-branch? You need to manually edit the README.md file to restore it to the state you want it to be in. In my case, it looked like this after the failed merge:

```
<<<<<<< HEAD
MAster edit This is a test
=======
SB edit This is a test
>>>>>>> second-branch
```

You need to remove all the parts you do not want. Once you've done that, make a commit.