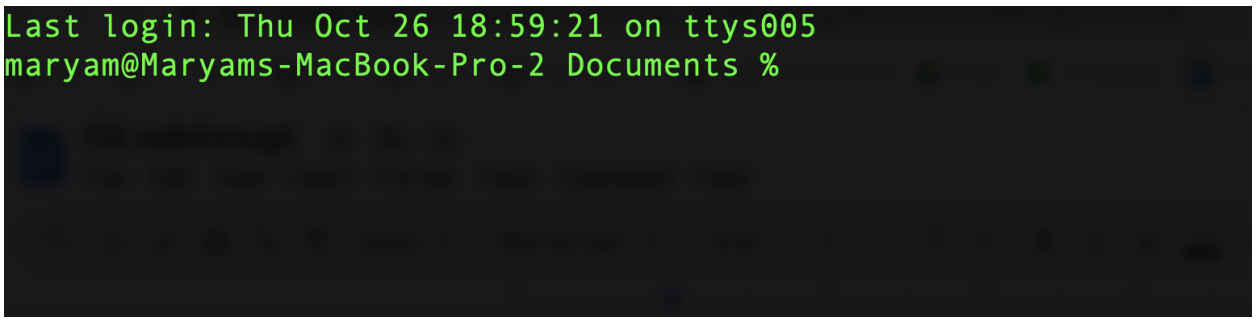
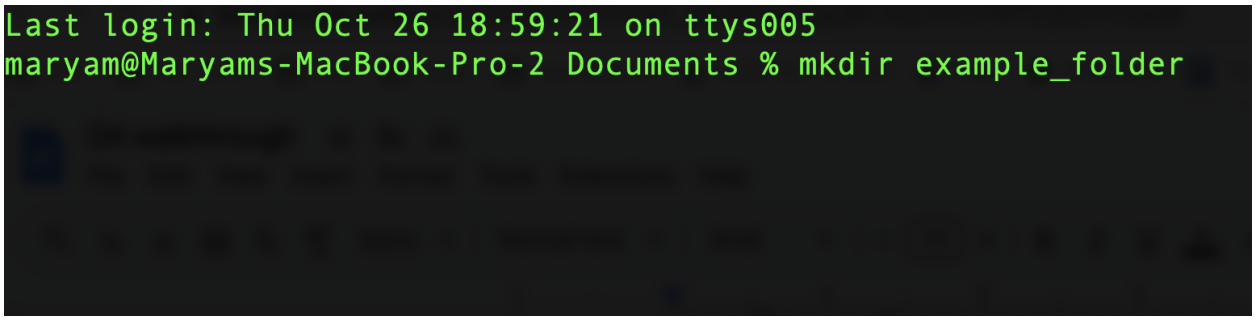
Creating a directory and some files

You wouldn't ordinarily do this if you were working on a real project- because the directory with all the files you wanted to track would already exist. These steps are purely to 'set us up' for this learning exercise.

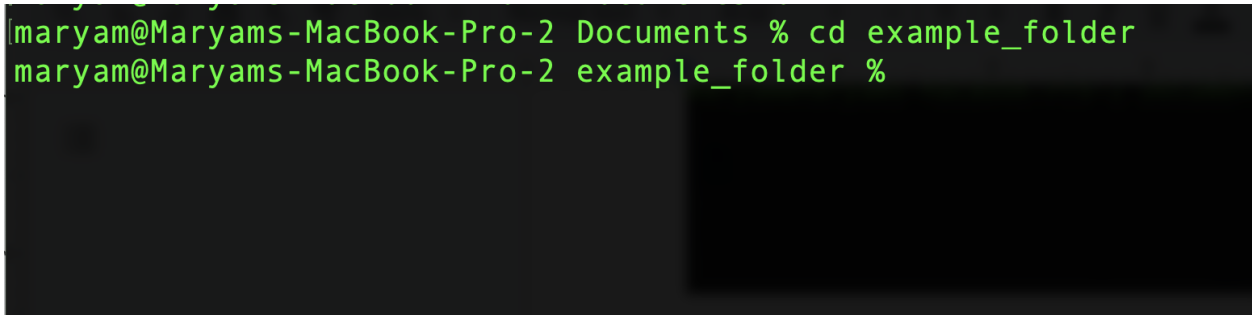
First off, we open up our terminal (this is sometimes called the command line, or the bash terminal, or just bash). It will look something like this:



Next up, we create a new directory (remember, this is just to set us up for the exercise). We can do this using the **mkdir** command:

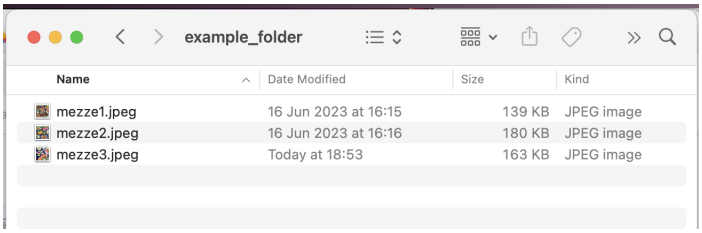


We then 'navigate' in the command line to this new folder we've created. We do this using the **cd** command:



You'll notice that before the % sign you can now see **example_folder** where before it said **Documents**. This shows us that we've successfully moved from our Documents folder to our new folder.

Next, add some files to your newly created folder. In this case, I've just added some images by copying and pasting some random files from my computer:



We're now all set up to begin version control!

Initialising a git repository

Let's imagine that we're very worried about accidentally deleting or corrupting these image files. We want to start tracking and taking snapshots of these files using git.

First of all, we need to tell git to start tracking **example_folder**. This is called initialising a git repository. A repository is simply a folder that we've asked git to start tracking.

We do this using the **git init** command.

We get a lot of information back when we run this command! The important thing is the last line, which tells us that a git repository has been created in our **example_folder**.

```
maryam@Maryams-MacBook-Pro-2 example_folder % git init
hint: Using 'master' as the name for the initial branch. This defaul
t branch name
hint: is subject to change. To configure the initial branch name to
use in all
hint: of your new repositories, which will suppress this warning, ca
ll:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk'
and
hint: 'development'. The just-created branch can be renamed via this
 command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /Users/maryam/Documents/example_
folder/.git/
```

Staging or adding files

It's not enough to simply initialise a git repository. We need to tell git exactly which files we want to take a snapshot of.

This is called staging or adding files. We do this using the **git add** command. We run this command for each file we want to stage.

```
maryam@Maryams-MacBook-Pro-2 example_folder % git add mezze1.jpeg
maryam@Maryams-MacBook-Pro-2 example_folder % git add mezze2.jpeg
maryam@Maryams-MacBook-Pro-2 example_folder % git add mezze3.jpeg
maryam@Maryams-MacBook-Pro-2 example_folder %
```

If we want confirmation that the files have been staged successfully, **git status** will show us this.

```
maryam@Maryams-MacBook-Pro-2 example_folder % git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   mezze1.jpeg
        new file:   mezze2.jpeg
        new file:   mezze3.jpeg

maryam@Maryams-MacBook-Pro-2 example_folder %
```

Making a commit

Finally, we're ready to take a snapshot. We do this using **git commit**. We also have to include a commit message, which is a note to ourselves explaining what is special or significant about this snapshot.

Because this is our first snapshot/commit in this folder, our commit message could say something like "initial commit after adding image files".

```
maryam@Maryams-MacBook-Pro-2 example_folder % git commit -m "Initial
 commit after adding image files"
[master (root-commit) 7485657] Initial commit after adding image fil
es
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 mezze1.jpeg
 create mode 100644 mezze2.jpeg
 create mode 100644 mezze3.jpeg
maryam@Maryams-MacBook-Pro-2 example_folder %
```

We get confirmation right away that these files have been included in a snapshot. Hooray!

Rolling back

Now imagine your cat has walked all over your keyboard while you're away from your desk, and has deleted one of your very important image files. Replicate this scenario by deleting mezze3.jpeg (you can do this using the point and click method in your file explorer).

How do we get our image file back?!

The first thing we need to do is take another snapshot of the current state of our repository. We can combine the process of staging and committing using this command:

git commit -am "my cat has accidentally deleted mezze3.jpeg"

The **-a** part of this command tells git that we want to stage/add all files that we previously staged/added and then make a commit. This saves us the hassle of typing **git add** for each of our files one by one.

(Important: if you're copying and pasting from this google doc into your terminal, you need to delete and re-type the speech marks in any commands you write, because they're not copied over to the command line correctly)

```
maryam@Maryams-MacBook-Pro-2 example_folder % git commit -am "my cat
 has accidentally deleted mezze3.jpeg"
[master 80a39d9] my cat has accidentally deleted mezze3.jpeg
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 mezze3.jpeg
maryam@Maryams-MacBook-Pro-2 example_folder %
```

Now, we can roll back to our first snapshot. We first run **git log** which shows us a list of all the snapshots/commits we've made so far, together with their unique identifiers (the long list of characters in yellow) and their commit messages.

```
maryam@Maryams-MacBook-Pro-2 example_folder % git log
commit 80a39d9858a344bb3084d1c4f22bd9c6b3faa13e (HEAD -> master)
Author: Maryam <maryam.oxford@gmail.com>
Date:   Thu Oct 26 19:12:39 2023 +0100

    my cat has accidentally deleted mezze3.jpeg

commit 748565713b7b5015875758cf2062421538216b18
Author: Maryam <maryam.oxford@gmail.com>
Date:   Thu Oct 26 19:09:01 2023 +0100

    Initial commit after adding image files
maryam@Maryams-MacBook-Pro-2 example_folder %
```

We do this by double clicking on the **identifier/hash associated with the snapshot we want to roll back to** and then hitting control+c or command+c to copy it. Then we write

git checkout [the hash of the commit we want to roll back to]

You'll see quite a long message like this. But, if you open up your file explorer you'll see that **mezze3.jpeg** (or whichever file you deleted) has magically reappeared!

```
maryam@Maryams-MacBook-Pro-2 example_folder % git checkout 748565713
b7b5015875758cf2062421538216b18
Note: switching to '748565713b7b5015875758cf2062421538216b18'.

You are in 'detached HEAD' state. You can look around, make experime
ntal
changes and commit them, and you can discard any commits you make in
 this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you
 may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead
to false

HEAD is now at 7485657 Initial commit after adding image files
maryam@Maryams-MacBook-Pro-2 example_folder %
```