**HOW DOES (G)IT WORK?**

* Git is a store/map of keys and values.
* It translates everything into “SHAWN” or SHA1 keys.
* SHA1 keys are 20 bytes in a hexadecimal format, they are a sequence of 40 hex digits.
* These are what GIT uses to store content in its map.

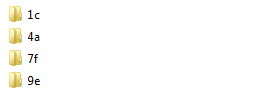
Using ‘git hash-object’ (this is not the full command) git will generate a SHA1 key for the desired object.

If the object is identical, for example the **exact** string ‘Apple Pie’, the SHA1 key will always be identical also.

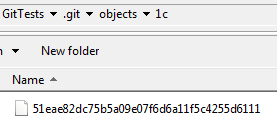
***EVERYTHING*** in GIT has a SHA1 key, this includes every branch, file, comment, readme – everything.

Using ‘git cat-file <SHA1key> -p’ we can read the file (assuming it is a text object) from the target SHA1 key.

GIT stores the SHA1 keys in the folder “objects” in the .git repository. They are stored in folders represented by the first 2 digits like below. This is done for organization purposes.



In each folder is a file with a name represented by the remaining 38 digits. We will use the folder ‘1c’ as an example.



In 1c there is a file which name is the remaining 38 digits of the 40 digit SHA1 key, to get the full key we add 1c to the front of this file name.

‘1c51eae82dc75b5a09e07f6d6a11f5c4255d6111’

If we run the previously mentioned command on this SHA1 key we get the following.

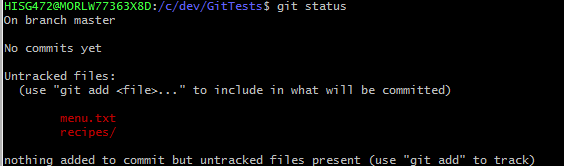


It contains 1 “blob” of data, which is a single file (or rather the data contained in the single file) and it contains 1 “tree” which a directory.

It is important to note that these commands are very low level and not needed to be learned at an early stage, this is just to demonstrate how GIT works.

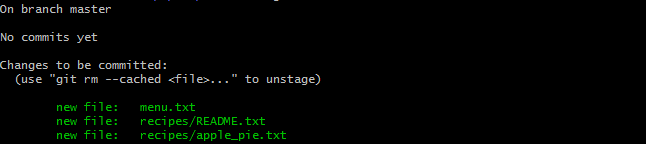
**COMMITING A PROJECT**

First we want to check the status of our project using “git status”.



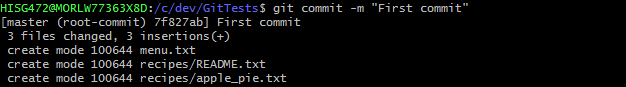
We have 1 untracked file and 1 untracked folder. The folder is marked with ‘/’ at the end.

At the bottom of this it suggests what we should do next ‘git add’, if we use ‘git add .’ it will add every file, alternatively you can use ‘git add menu.txt’ to add just the menu. After typing this nothing will happen on your console we need to enter ‘git status’ again to see changes.

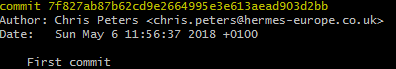


We have added the text file and the folder with all of its content; we are “staging” these files for committing, which is what we’ll do next. By typing ‘git commit -m “First commit” ’.

The -m is for message, followed by the message we want to leave with our commit. In this case it is “First commit”.



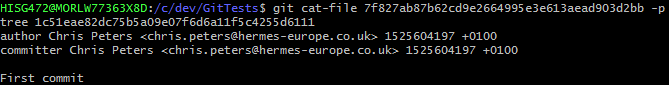
If we do ‘git log’ now:



We can see the commit SHA1 key along with other details including the message we wrote.

We can now do ‘git cat-file 7f827ab87b62cd9e2664995e3e613aead903d2bb –p’

This shows us the contents of the SHA1 object.



Now if we go even deeper and do the same command on the tree SHA1 key we can see.



A side note here is you do not need to use the full SHA1 key, as long as you provide the first group GIT will figure out which you mean, be sure to include more than 2 or 3 as if the first 2 or 3 digits are the same it will not work.

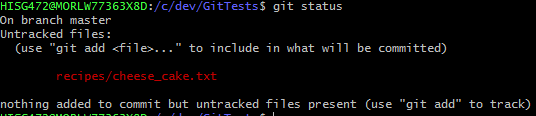
Now let’s go **even deeper** and run the same command on the blob SHA1 and the tree SHA1 keys.



The first one reads the menu.txt file and the second one reads the recipe folder and displays the files.

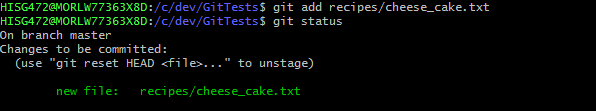
***THE SECOND COMMIT***

So we’re going to add another recipe to our recipe folder, however we’re lazy and we haven’t updated our menu.txt file yet. I have changed this file so let’s see what ‘git status’ tells us now.



It shows we have a new untracked file. We will now “stage” this file for committing by using the command ‘git add recipes/cheese\_cake.txt’.

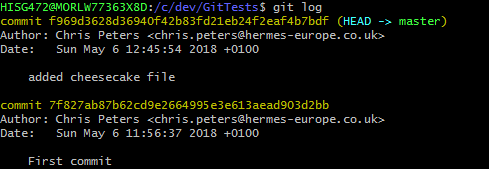
Again, the console will not show anything new until we do another ‘git status’.



This shows we have stages our new file. We now need to commit using ‘git commit -m "added cheesecake file" ’

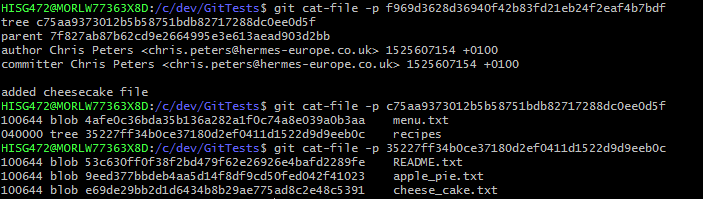


Now let’s do ‘git log’.

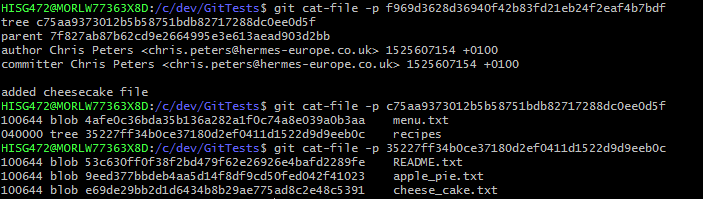


It shows or current (HEAD) commit and the previous commit with other details and the messages we wrote.

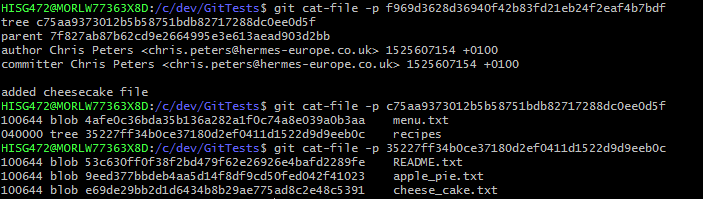
Now let’s ‘cat-file’ a bunch more to see the differences, we will take the same route as the first commit.



The first difference being that we now have a parent SHA1 key, every commit will have a parent other than the very first commit.



The recipes tree SHA1 is different because we added a new folder, changing the way the SHA1 key is generated, meaning a different key is now relevant to that object.

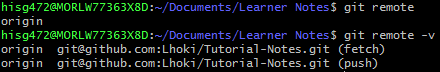


The README.txt and apple\_pie.txt SHA1 keys are the same because nothing has changed but we do have an additional SHA1 key for the new file we created.

**REMOTES REPOSITORIES**

A remote repository is a version of your project hosted on the internet. Collaborating with others involves being able to use remotes properly, this involves pushing and pulling data from them from several different sources and being able to manage the data correctly.

To see your remotes you can type ‘git remote’ the default is “origin” – git will create one for you when you ‘git clone’ a project. You can also type ‘git remote –v’ to see the URLs of the remotes which makes pushing and pulling to projects you’ve already set up very easy.



We can add remotes by using the command ‘git remote –add <name> <url>’.

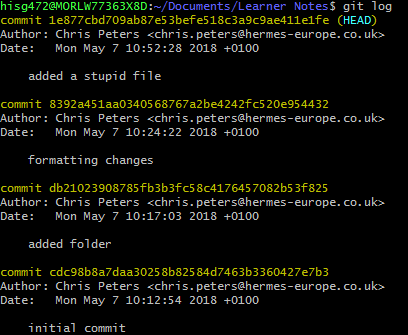


Now you can use origin in place for push, fetch and clone.

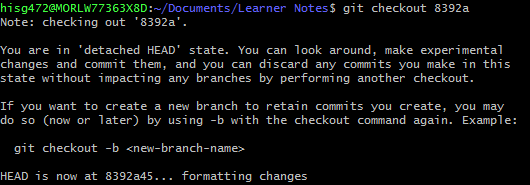


**GIT CHECKOUT**

‘git checkout’ can be used to view older commits and potentially reset your work to an earlier version if you need to. Start by doing a ‘git log’.



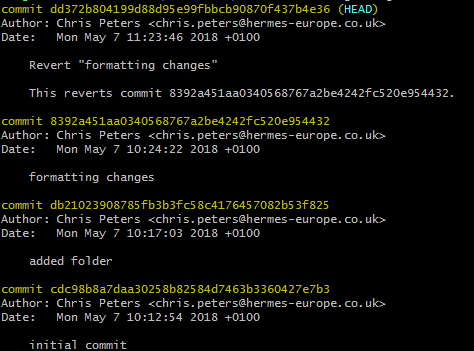
You can see here I have accidently added a stupid file, I will now do ‘git checkout 8392a4’ to view my older commit.



The next command is ‘git revert’.



Followed by another ‘git log’ to verify we’ve done what we wanted to.



As you can see, we have moved the “head” to another commit, this commit reverts to the commit we specified.

**GIT COMMANDS**

git init *//Makes the current folder a git directory.*

ls *//View files.*

ls al *//View all files, including hidden.*

cd <directory> *//Jumps into directory.*

cd .. *//Goes up one level from the current directory.*

cat <filename>.txt *//Reads the text file.*

cat <directory>/<filename>.txt *//Reads the text file in the directory without changing directories.*

tree . *//Shows the filepath as a tree (DOESN’T WORK IN WINDOWS).*

git log *//Shows the commit history.*

git remote *//Shows the remote repository names, the default is origin.*

git remote –v *//Shows the remote URLs for push and fetch.*