Git

Git is a distributed Version Control System and allows us to

First, it works great with tracking changes. You can

• Go back and forth between versions

• Review the differences between those versions

• Check the change history of a file

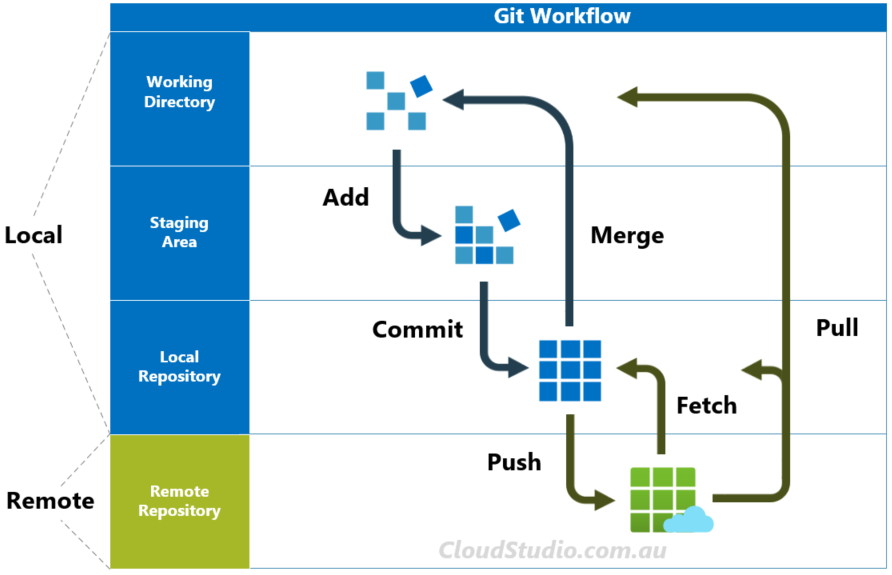
• Tag a specific version for quick referencing

Git is also a great tool for teamwork. You can

• Exchange “changesets” between repositories

• Review the changes made by others

Github is the hosting service that allows us to store git repositories in the cloud.

Git Workflow

The main feature of Git is its “Three States” system. The states are the working directory, the staging area, and the git directory:

• The **working directory** is just the current snapshot that you are working on such as python file or html file.In here you modify and edit the files on the Working Directory in your text editor.

• The **staging area** is where modified files are marked in their current version, ready to be stored in the database. In here You put the files you want to record the current state on the Staging Area.

• The **git directory/Local repository/commit history** is the database where the history is stored Files that are here are saved point.In here You take a snapshot of the project with a commit.

So, basically Git works as follows: you modify the files, add each file you want to

include in the snapshot to the staging area (git add), then take the snapshot and add

them to the database (git commit). For the terminology, we call a modified file added

to the staging area “staged” and a file added to the database “committed.” So, a file goes

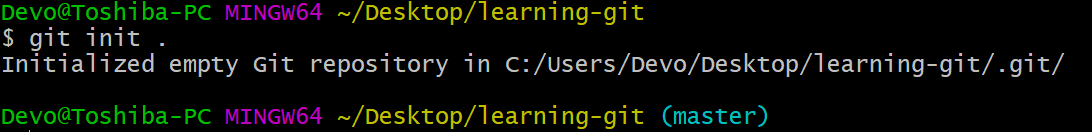
from “modified” to “staged” to “committed.”

From here you can send the local repository to a remote repository such as github using git push.

# Git init

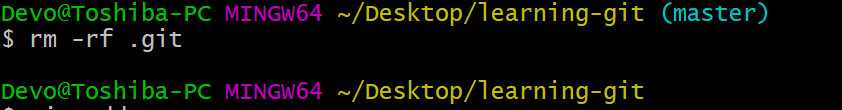
Initialize working directory

* While in your working directory use the command **“git init .”**
* This tells git that the working directory is a git a repository.
* A git repository is a collection of files of various different versions of a project.
* When you run git init, a folder called .git/ is created. Within this folder, you will find a list of files that relate to the branches of a repository,and other crucial pieces of information about Git.
* The .git/ folder is hidden. This is because the folder begins with .git/. This folder is hidden to protect you from accidentally deleting its contents.If we use the ls command , we can see this folder
* Once the working directory is initialised it can be used into the git workflow (to git add, commit etc.)
* This is used for brand new projects. Not existing projects.Cloned repositories are accompanied with all the configuration information you need.



Ex. Of using git init. We use init in the working directory we want. We know when it is initialized when we become (master)

To uninitialise a working directory We can remove the git repository using **rm -rf .git/** (we know it is unutilized because there is no longer a master)

****

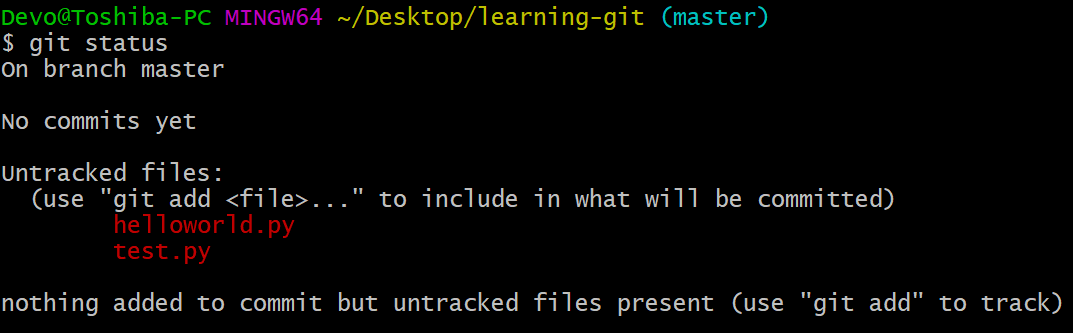
# Git Add

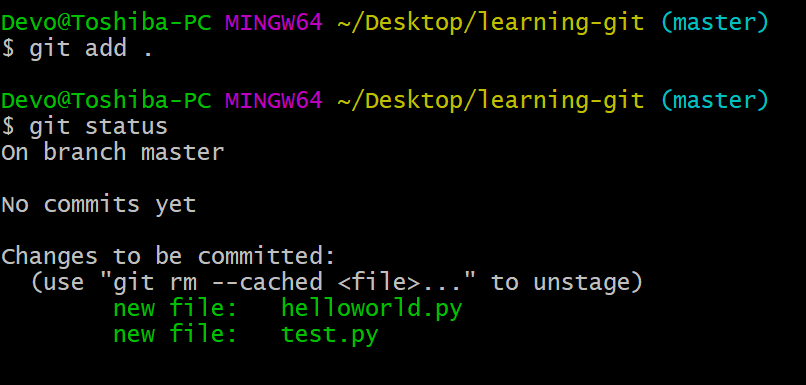
* After your working directory has been initialised thus creating a local git reporirsty, you can now add files to the staging area.
* To add files to the staging area use the command **git add**

| Command | Description |
| --- | --- |
| **git add -A** | Adds all (new, modified, deleted) files to Stage |
| **git add .** | Adds all (new, modified, deleted) files in current folder to Stage |
| **git add ‘file name’** | Adds that specific file to the staging area |
| **git add --ignore-removal .** | Adds to stage new and modified files only |
| **git add -u** | Adds to stage modified and deleted files only |

To unstage a staged file use rm --cached ‘file name’ or rm –r –cached .

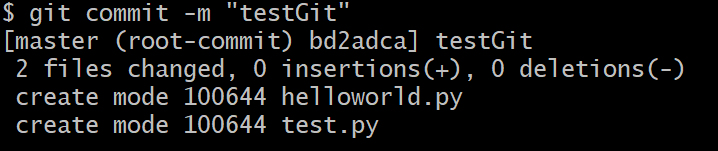
## Git Status

* **git status**
* Shows the current state of the working directory and the staging area.
* It indicates which files are added to the staging area and those that are not.

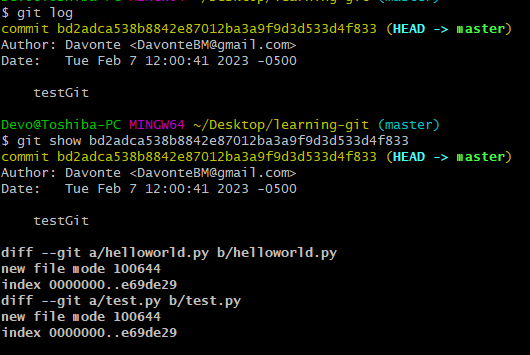


# Git commit

* A commit is just a snapshot of the entire project at a certain time. A save point
* The "commit" command is used to put your files from the staging area to the local repository.
* A commit is identified by its name, a 40-character string that is obtained by hashing the commit. It is a simple SHA1 hash so multiple commits with the same information will have the same name.

**git commit -m “Message”**

## How to see the changes of files that are committed

1. git log (The git log command displays committed snapshots. It lets you list the project history, filter it, and search for specific changes.) 
2. copy the hash given from the git log
3. git show “hash”

**git diff** shows the changes between the working directory and what has been committed

To discard changes of files that were changed in the working directory (not staged or committed) use **git restore**

# Git Push

Git push adds our local existing committed repository to a remote repository such as github.

To take an existing project you are working on and push it, so it also exists on GitHub follow the following steps.

1. Go on your github.com account.
2. Create a new repository on github.
3. Enter the following git commands;

* git remote add origin [git@github.com](mailto:git@github.com):DavonteBM/”GithubRepsorityName”.git (where GithubRepsorityName is the name that you created in step 2)
* git branch -M main
* git push -u origin main

1. When ever stuff has been committed in your local repo, you can push it with the following command

* git push

SSH Key (optional)

(If this is using a different computer set up an SSH for that computer. Only one is neeed per device)

4. Set up SSH key

1. Generate SSH key

Using git bash, paste the text below, substituting in your GitHub email address.

* ssh-keygen -t ed25519 -C "your\_email@example.com"

1. Adding your SSH key to the ssh-agent

Type into git bash the following

* eval "$(ssh-agent -s)"
* ssh-add ~/.ssh/id\_ed25519

1. Add SSH key to github

Type into git bash.

* clip < ~/.ssh/id\_ed25519.pub (This copies the ssh key to the clipboard)
* Go to github.com>settings>ssh key
* Click new sshkey.
* Ctrl v or paste the sshkey clipped from the last step into the form.
* Create ssh button

# Git Pull

Git pull is the reverse of git push, in that git pull pulls from a remote repository like github and adds it to our local machine as an already intialized working directory.

When working as a team on a project, it is important that everyone stays up to date you can do this by doing git pull.

To pull from a remote repo to your local repo (that you have already pushed to) use this command

**git pull**

# Branches

In Git, a branch is a new/separate version of the main repository.

Branches allow you to work on different parts of a project without impacting the main branch.

When the work is complete, a branch can be merged with the main project.

Create a Branch

To create a branch do the following command

git branch <nameOfBranch>

(where nameOfBranch is the name you want to call the branch)

Check the branches available

To check the available branches available on our local machine, do the following command

git branch

(the branch name with a \* is the branch that you currently are in)

To get a list of remote branches do the this command

git branch -r

Change Branches

To change the branch that you currently are on do

git checkout <nameOfBranch>

(where nameOfBranch is the branch you want to be in)

**Delete a branch**

**To delete a branch, checkout into the main branch. Then type the following command**

**git branch -d <localBranchName>**

Merge a Branch

After committing inside a branch, you can merge it into the main branch.

To do this, checkout into the main branch. And do the following command

git merge <nameOf branch>

(Where nameOf branch is the branch that you want to merge with the main branch)

When working for a company or a project. You will never push your local branch to the main branch

Instead you will create a branch, commit your work to that branch.

Then push that branch to a remote repo like github.

Lately, you would issue a pull request which would then be reviewed and merged by other developers.

m