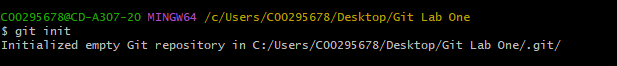
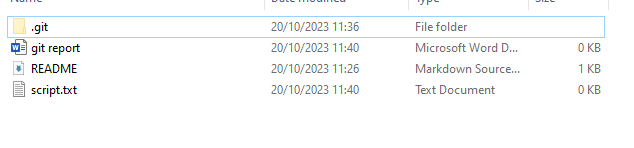
# Init

Git init is used to initialise a git repository



It creates the repository in the folder directory GitBash was opened in

It also creates a new folder as a hidden item



This .git folder indicates that this is the folder where the repository was created.

Syntax:

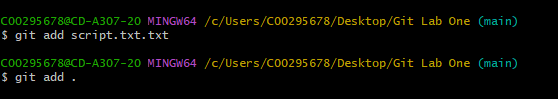
Git init

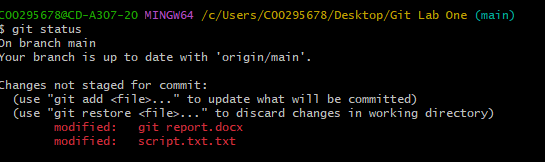
# Add

Git add sends files to the “staging area”. In this area, they are prepared to be committed. Add should be used anytime a file has been updated by the user and hasn’t been updated in the repository. This includes any new files introduced, and any previous files that have been updated. The user can specify any file within their folder by typing the filename after “git add”, and they can add all files in the folder by using a “.”

Syntax:

Git add <filename> / .



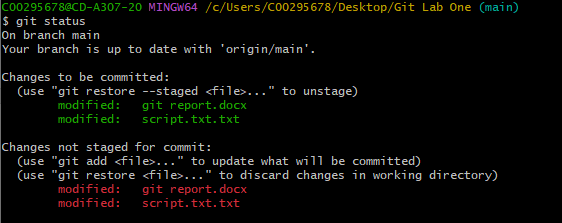


# Status

Git status is used to display the current status of files in the source folder and the staging area. It will display what files are yet to be added and which files are yet to be committed. A red file is one that hasn’t been added to the staging area yet, while green files are files that exist in the staging area, but are yet to be committed. Git status should be used following any add or commit done by the user to verify the process has been completed correctly.

Syntax:

Git status



# Commit

Git commit is used to fully introduce a file from the staging area into the repository. As stated previously, a git status will show uncommitted files as green. Committed files won’t appear when a git status is called, a signal that the commit was successful. Unliked an add, however, a commit must come with a message. The line must read “git commit -m “<commit title>”. This means that when the commit is viewed in the repository, there is a title included. If this is not done, git will open up VI, a program that is much harder to use than a “-m” function.

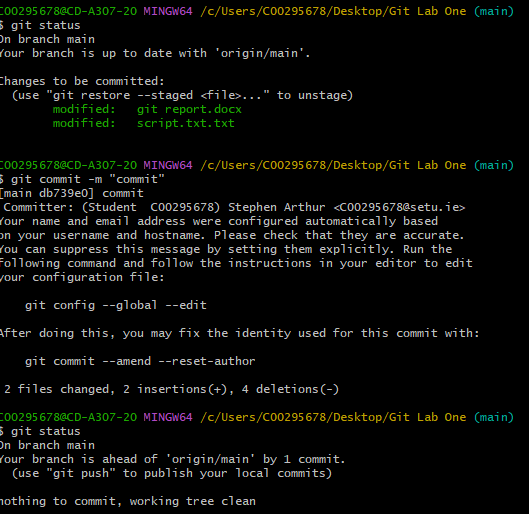
Syntax:

Git commit <filename>

Git commit .

Git commit <filename> -m “comment”

Example below:



# Remote

Git remote creates a remote version of a repository. This version of a repository is common among multiple coders, and will be held on an online host like GitHub, or a more private host for a large corporation. While the users will have a local repository to work on, they must push their changes up to the remote, even if those changes are committed to the local repository. A remote is added with the command “git remote add origin remote\_directory\_url” This version of the repository is updated by multiple users separately, and as such is used for working in a group.

Syntax:

Git remote add origin <repositorydirectory>

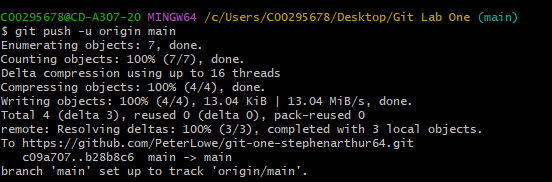


# Push

Git push is used to push up the local changes to a repository to the associated remote. The -u function must be used in order to associate the push request with the remote the changes are being pushed to. A push must always be done when git is being closed for the day, so as to protect the files being used. Pushes should also be done often so that, in the event of something like a crash, the minimum amount of work is lost.

Syntax:

Git push -u (associates) origin <branchname>



# Branch

Git branch creates a new branch in the local repository where files can be added. A branch is a string of nodes that acts parallel to the main or master branch. A new branch can be made with the “git branch <branchname>” command. This new branch can be swapped to using the checkout command, and can now add and commit files to this branch. When this branch is pushed to the remote, it will be prompted to merge with the main branch.

Syntax:

Git branch <branchname>



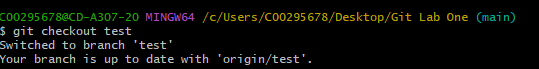
# Checkout

Git checkout is used to swap between nodes and branches. It can be used to change which node or branch is being updated with files. The command “git checkout <node or branch>” is used to swap the section being used. It can also be used to create new branches with “git checkout -b <branch name>”.

Syntax:

Git checkout <branchname>

Git checkout -b <branch name> (makes new branch)



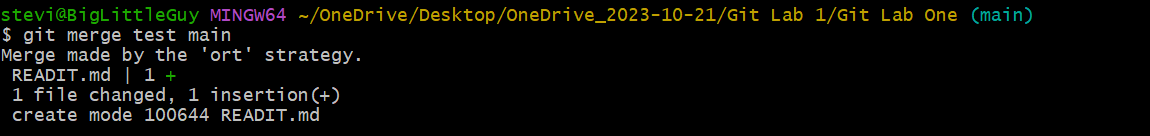


# Merge

The merge function in git is used to merge 2 branches together. This function is useful for re-incorporating parallel developments to a project but must be done with caution. If there is a conflict in the merge, like two identical variable names, the merge is stopped. The merge is done by git applying the commits done to the parallel branch to the branch being merged into. If conflict occurs, --abort walks back the merge, while --continue keep it going. This function is used often to ensure a common project. It must also be done regularly for small changes, to prevent a massive merge that results in far too many conflicts than can reasonably fixed.

Syntax:

Git merge <branch1> <branch2>



# Log

The log function is used to view the history of commits in the repository. It displays every commit that has been made. This display only shows a limited number of commits, so it is navigate by using the space key, and the user can quit using the q key. Variations like --stat show the individual changes made in each commit, while --graph displays the information in a more graphed format, and -n shows the last n commits. This function is useful for tracking previous commits, and is also one of the reasons the user must always name commits, so as to make the log clearer.

Git log

Git log --stat

Git log --graph

A computer screen with many white and yellow text

Description automatically generated

A computer screen shot of a computer program

Description automatically generated

A computer screen shot of a computer code

Description automatically generated

# Clone

Git clone creates a complete clone of a repository that can be independently updated and changed. It operates as its own repository. The clone can then be merged back into the main repository with a simple pull command. The clone function can be useful in the even that the user wants to make a series of changes to a project without interrupting the main, or if the user wants to work remotely on a machine apart from the codebase.

Syntax:

Git clone <repository>

Example below:

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

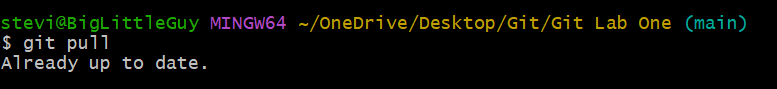
Description automatically generated

# Pull

Git pull is used to update a local repository with a remote repository. It pulls the commits made to the remote repository and applies them to the local. Options are then used to decide if these changes will rebase the local or merge with it. This command is used when the remote is ahead of the local, which may be useful if changes have been made to it and your local repository is behind. If the local is already caught up with the remote, nothing happens

Syntax:

Git pull <options> <repository>



# Stash

Git stash creates a stash of a dirty node, storing it away while the user can return to a clean version of the repository. The stash can be recovered at any time with the “apply” function. This could be useful if the user wants to temporarily move back to a clean repository without losing all of their work on the node. Stashes are also named as “stash@{n}”, n being ordered with 0 as the most recent, 1 as the next most recent, etc.

Syntax:

Git stash

Git stash apply

A screenshot of a computer program

Description automatically generated

# Rm

Git rm removes a file from either the working directory or staging area, depending on the syntax. This command should be used with caution, as there is no warning, it will simply remove the files. The user should be sure that they use this command carefully and only when they are sure they know what the file is and what they are removing it from.

Syntax:

Git rm

Git rm <filename> --cached -> removes from staging area

Rm <filename> -> removes from working directory

A computer screen shot of text

Description automatically generated