**Basic Git Workflow**

**Introduction to Git**

Git is a version control system, which allows you to track and manage changes to all types of files.

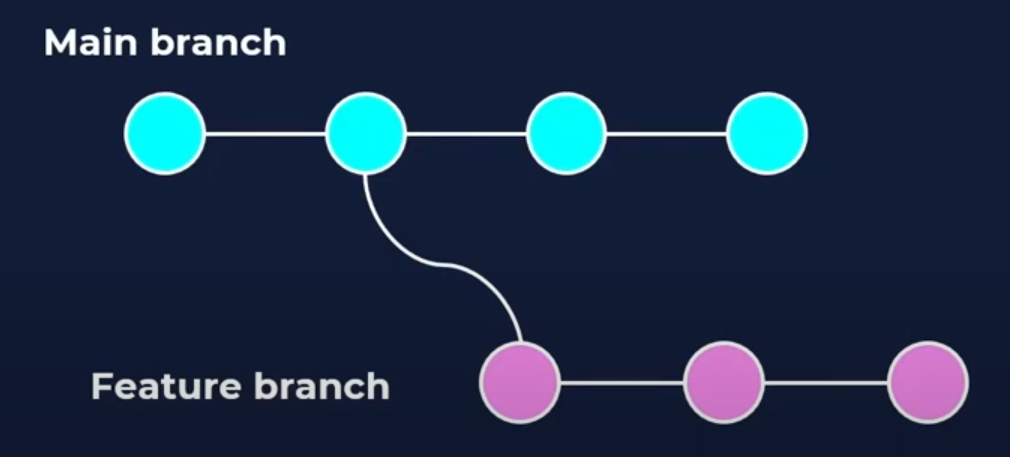
Its primarily used for allowing groups of developers to work collaboratively on large software projects, but can also be used by small teams or solo developers

This is accomplished by the evolution of set of files. Instead of managing each file individually and sharing it manually with your team, you can use a repository which contains a collection of files and version information for a project. This repository tracks all changes made to the files building history over time

**Advantages of Git**

One of the biggest advantages of Git is its branching capabilities. A branch is a way to keep a new feature of modification separate from the main project. It contains a different version of the files, which include any changes made after it was created from the project code.

As soon as a repository is created, a main branch is generated. From there, one can work on code and track their work with commits (a commit record changes to one or more files in a branch). However, another member can also branch out from the specific point in the repositories history and work on their code. This leaves the main project intact and isolated from any experimental code that someone’s working on:



Git is also very efficient for reviewing completed code. Once code is ready for review, a user can create what’s called a pull request. The pull request is essentially a dedicated forum for discussing proposed changes in a soon-to-be merged branch. Not only does this make it easier for a project leads to keep track of changes, but it also lets developers initiate discussions around the work before integrating it with the result of the code base

**Introduction to Github**

Github is a website and cloud-based service that hosts Git repositories. While Git is a command line tool, Github provides a web-based graphical interface with many of its own added features for easier navigation and collaboration with projects

How to use: <https://www.youtube.com/watch?v=eNw8MgguLsQ&t=42s>

**git init –** creates a new Git repository

The word init means ***initialise***. The command sets up all the tools Git needs to begin tracking changes made to the project



**Git Workflow**

A Git project can be thought of as having three parts:

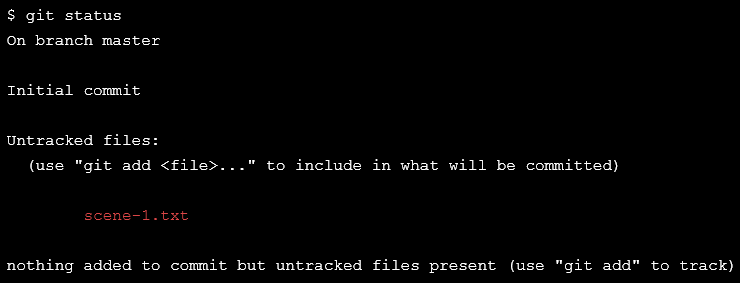
* *A Working Directory:* where you’ll be doing all the work: creating, editing, deleting, and organising files
* *A Staging Area:* where you’ll list changes you make to the working directory
* *A Repository:* where Git permanently stores those changes as different *versions* of the project



**git status –** inspects the contents of the working directory and staging area

As you write the screenplay, you will be changing the contents of the working directory. You can check the ***status*** of those changes with:





In the output, notice the file in red under *untracked files*.

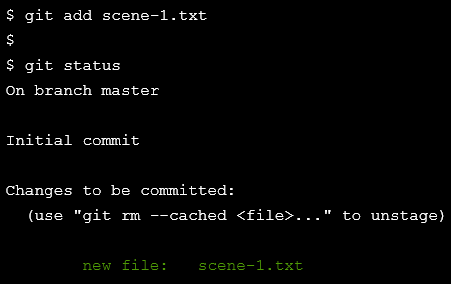
Untracked means that Git sees the file but has not started tracking changes yet

**git add –** adds files from the working directory to the staging area

In order for Git to start tracking *scene-1.txt*, the file needs to be added to the staging area. We can add a file to the staging area with:



The word *filename* here refers to the name of the file that you are editing, in this case, it would be *scene-1.txt*



In the output, notice that Git indicates the changes to be committed with *“new file: scene-1.txt”* in green text. Here, Git tells us the file was added to the staging area

**git diff –** shows the difference between working in the working directory and staging area

Image we type another line in *scene-1.txt*. Since the file is tracked, we can check the differences between the working directory and the staging area with:

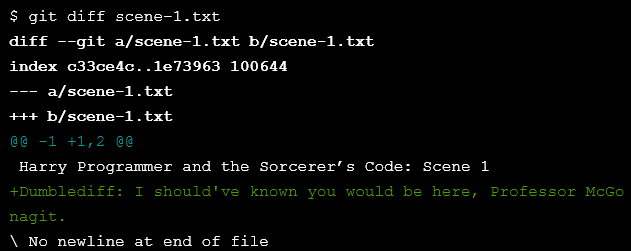


Here, *filename* is the actual name of the file. If the name of my file was *changes.txt*, the command would be:





Note: the first line was there before, and the second line is the line that we just added

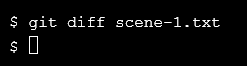


Notice the output:

* *“Harry Programmer and the Sorcerer’s Code: Scene 1”* is in the staging area, as indicated by the colour white
* Changes to the file are marked with a + and are indicated by the colour green



We can then add the changes to the staging area, so when you type in…



…you get no result, as there is no difference between the working directory and the staging area

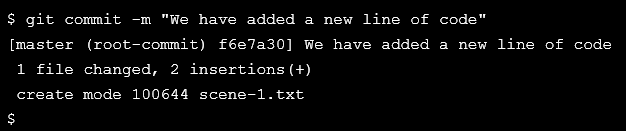
**git commit –** permanently stores the file changes from the staging area in the repository

A ***commit*** permanently stores changes from the staging area inside the repository. However, before we use ***git commit***, one more bit of code is needed for a *commit*: the option ***–m*** followed by a message. Here is an example:



Standard Conventions for Commit Messages:

* Must be in quotation marks
* Written in the present tense
* Should be brief (50 characters or less) when using ***–m***



**git log –** shows a list of all previous commands

Often with Git, you’ll need to refer back to an earlier version of a project. Commits are stored chronologically in the repository and can be viewed with:





In the output, notice:

* A 40-character code, called SHA, that uniquely identifies the commit. This appears in orange text
* The commit author (you!)
* The date and time of the commit
* The commit message