**Hello Git**

Git is a software that allows you to keep track of changes made to a project over time. Git works by recording the changes you make to a project, storing those changes, then allowing you to reference them as needed.

We’ll learn Git by using it to help us write a screenplay called *Harry Programmer and the Sorcerer’s Code*.

**Instructions**

**1.**

We’ll get started by taking a look at the screenplay project.

In **scene-1.txt**, add this text:

Harry Programmer and the Sorcerer’s Code: Scene 1

Then press enter to create a new empty line. Once you’ve created the new line, click Run.

**git init**

Now that we have started working on the screenplay, let’s turn the **sorcerers-code** directory into a Git project. We do this with:

git init

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

**Instructions**

**1.**

In the terminal, initialize a new Git project.

Notice the output:

Initalized empty Git repository in /home/ccuser/workspace/sorcerers-code/.git/

The Git project was created. Click Next to continue.

# Git Workflow

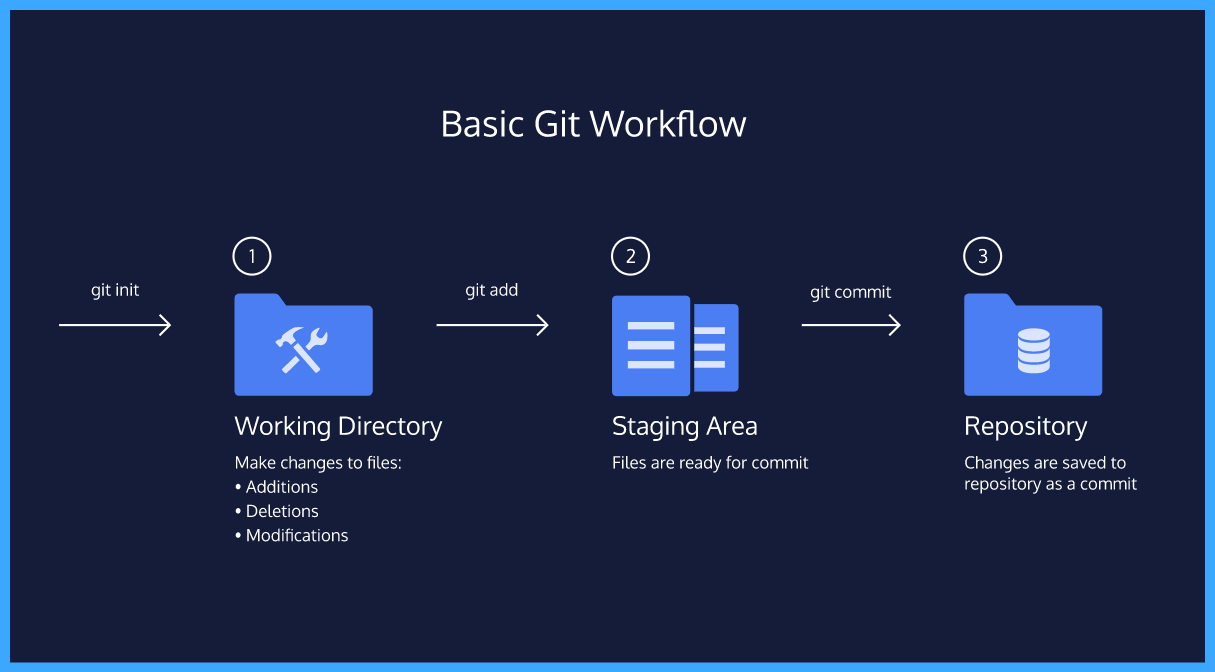
Nice! We have a Git project. A Git project can be thought of as having three parts:

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

The Git workflow consists of editing files in the working directory, adding files to the staging area, and saving changes to a Git repository. In Git, we save changes with a commit, which we will learn more about in this lesson.

**Instructions**

Take a look at the diagram. Before we move on, it will help to be familiar with the three parts of the Git workflow. Click Next to continue.



# git status

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

git status

**Instructions**

**1.**

From the terminal, check the status of the **sorcerers-code** project.

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.

Hint

If you’re having trouble accessing the terminal, try refreshing the webpage.

# git add

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:

git add filename

The word filename here refers to the name of the file you are editing, such as **scene-1.txt**.

**Instructions**

**1.**

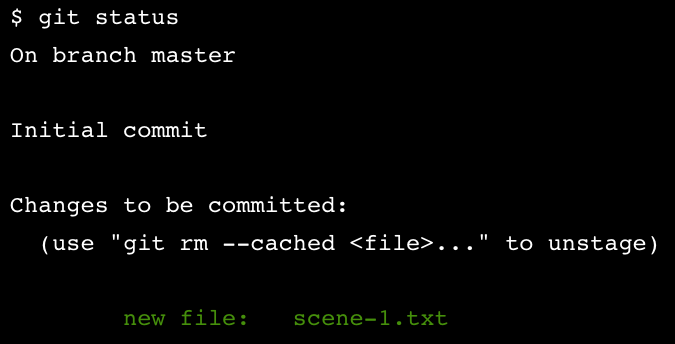
Add **scene-1.txt** to the staging area in Git. Recall that you will need to identify the file by its name.

**2.**

Check the status of the project in Git.

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.

Hint

The output of git status should look similar to this:

**git diff**

Good work! Now you know how to add a file to the staging area.

Imagine that 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:

git diff filename

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

git diff changes.txt

**Instructions**

**1.**

In the code editor, add this text to **scene-1.txt**:

Dumblediff: I should've known you would be here, Professor McGonagit.

Click Run.

**2.**

From the terminal, use the new command to check the difference between the working directory and the staging area.

Notice the output:

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

Stuck? Get a hint

**3.**

Add the changes to the staging area in Git. Recall that you will need to identify the file by its name.

Hint

We can add a file to the staging area with:

git add filename

**git commit**

A *commit* is the last step in our Git workflow. A commit permanently stores changes from the staging area inside the repository.

git commit is the command we’ll do next. However, one more bit of code is needed for a commit: the *option* -m followed by a message. Here’s an example:

git commit -m "Complete first line of dialogue"

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

**Instructions**

**1.**

Make your first commit! From the terminal, type the command along with a commit message. The message should describe the point of the commit.

If you’re having trouble thinking of a good commit message, reflect on how the project has changed since it began.

**git log**

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:

git log

**Instructions**

**1.**

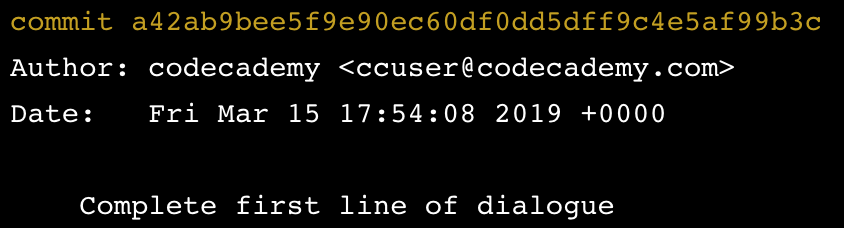
From the terminal, log a list of your commits.

In the output, notice:

* A 40-character code, called a *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

Click “Next” to continue.

Hint

Here’s what an example log looks like:

**Generalizations**

You have now been introduced to the fundamental Git workflow. You learned a lot! Let’s take a moment to generalize:

* Git is the industry-standard version control system for web developers
* Use Git commands to help keep track of changes made to a project:
  + git init creates a new Git repository
  + git status inspects the contents of the working directory and staging area
  + git add adds files from the working directory to the staging area
  + git diff shows the difference between the working directory and the staging area
  + git commit permanently stores file changes from the staging area in the repository
  + git log shows a list of all previous commits