# Introducing the Git Repository

## Initializing a repository

A repository is how we refer to a project version controlled by Git. In reality, a repository is represented by a hidden directory called .git that exists within a project directory and it contains all the data on the changes that have been made to the files in a project.

To turn a project directory into a Git repository we must initialize the repository. Initializing a repository simply means creating a repository.

When we initialize our repository the .git directory is going to be created inside our project directory. Because the .git directory is a hidden directory we won’t be able to see it unless we explicitly make hidden files visible.

[ WARNING ]

You should never touch any files or folders inside your .git directory. Doing this could have undesirable consequences on your repository. You should never delete this directory unless you want to delete your repository.

[ FOLLOW ALONG 2-1 ]

Make hidden files visible in your system.

Now look at the contents of the rainbow project directory in the file system window. There should be no visible or hidden files.

rainbow $ ls -a

What to notice:

In step 2, we see that the rainbow project directory is empty in the file system.

In step 3 we see that the rainbow project directory is empty in the command line.

We illustrate the above observations in Visualize It 2-1.

**[ VISUALIZE IT 2-1 ]**

A picture containing screenshot, rectangle, design

Description automatically generated

The rainbow project folder is empty.

To initialize a Git repository we are going to use the git init command. Our location in the command line has to be at the project directory we want to turn into a repository when we execute this command.

[ SAVE THE COMMAND ]

*git init*

Create a git repository

It’s time to turn our rainbow project directory into a Git repository, off to Follow Along 2-2.

[ FOLLOW ALONG 2-2 ]

To see the .git directory being created, make sure to have a view of the contents of your rainbow project directory in your file system.

rainbow $ **git init**  
Initialized empty Git repository in /Users/annaskouikari/desktop/rainbow/.git/

Go to the rainbow project directory in the file system and look at the .git directory that was just created. Open the .git directory to view the contents inside.

What to notice:

* Git created the .git directory inside the rainbow project directory. We illustrate this in Visualize It 2-2.

|  |
| --- |
| * [ VISUALIZE IT 2-2 ] |

A picture containing text, screenshot, diagram, font

Description automatically generated

|  |
| --- |
| We initialized our repository. |

We turned our rainbow project directory into a Git repository. Inside the .git directory there are various files and folders. Some of them represent the areas of Git that we are going to learn about next and for which we will add a representation in our visual model. As we continue working on our project, more files and folders will be created in our .git directory and we will continue learning about some of these as we go along.

In Figure 2-1, we can see an example of the contents of a .git directory in a newly initialized repository.

A screenshot of a computer

Description automatically generated

Now, the first area for which we will add a representation in our visual model is the .git directory itself which represents the repository as seen in in Visualize It 2-3.

**[ VISUALIZE IT 2-3 ]**

A computer screen shot of a folder

Description automatically generated with low confidence

A representation of the repository in our visual model.

Next, let’s continue building our visual model and learning about the different areas when working with Git.

## **The areas of Git**

There are three additional areas that are important to be aware of when we are working with Git:

* Working directory
* Staging area
* Commit history

Next we will cover what each of these are, how they relate to one another, and we will continue building our helpful visual model.

Let’s start off with the working directory.

## **Introducing the working directory**

The working directory is represented by the contents of our project directory (not including the .git directory). It is sort of like a workbench. It contains all the files and folders of one version of our project. And it is where we add, edit, and delete those files and folders.

To go over an example, let’s consider our example book project in Example Book Project 2-1.

###### EXAMPLE BOOK PROJECT 2-1

Suppose our book has ten chapters and consists of ten files, one for each chapter.

To add each of these chapter files to our project we would create these files in our working directory.

If we wanted to make any changes to the content of those chapters we would start by editing the files in the working directory.

And finally, if we decided we wanted to remove an entire chapter of our book, then the first step would be to delete the corresponding file in the working directory.

So, as we can see in the example above, the working directory is really where we make all the modifications to the content of a project.

Now, let’s return to our rainbow project, in our case, the working directory is represented by the contents of the rainbow project directory.

To continue building our visual model we will add a representation for the working directory to it, which is illustrated in Visualize It 2-4.

**[ VISUALIZE IT 2-4 ]**

A picture containing screenshot, text, line, diagram

Description automatically generated

Adding a representation for the working directory in our visual model.

[ NOTE ]

In our file system, we can see that the .git directory which represents the repository is inside the rainbow directory which represents the working directory.

In our visual model, however, the rectangle representing the repository is not inside the rectangle representing the working directory.

We designed the visual model like this on purpose in order to show that the working directory and repository operate independently of one another in some ways. This visual model will allow us to see how these two areas fit into our Git workflow in different ways.

However, even though we make this distinction between the working directory and the repository, it is important to know that when people refer to a project version controlled with Git they will refer to the project folder as the repository.

For example, in our case, we would refer to the rainbow repository.

Now that we have created a representation for the repository and the working directory in our visual model, let’s take a closer look at the repository.

Within a repository there are two important areas we want to explore further, the staging area and the commit history. Let’s first discuss the staging area.

## **Introducing the staging area**

The staging area is like a rough draft space. It is where we can add and remove files, when we are preparing what we want to include in the next saved version of our project (our next commit), in order to be able to explicitly craft what will be included. The staging area is represented by a file in the .git directory called index.

[ NOTE ]

The index file is only created if we have added at least one file to the staging area in our project. In the rainbow project directory, we have not yet added any files to the staging area, therefore the index file is not yet visible in the .git directory. In Chapter 3, we will add a file to the staging area and we will see the index file being created.

We will learn more about why the staging area is so helpful when we practice adding a file to it in Chapter 3. For now, we will add a representation for it inside the repository in our visual model in Visualize It 2-5.

**[ VISUALIZE IT 2-5 ]**A picture containing screenshot, rectangle, text, diagram

Description automatically generated

Our visual model now includes a representation for the repository, the working directory, and the staging area.

The final area we want to introduce is the commit history. But before we do so let’s properly introduce what is a commit.

## **What is a commit?**

A commit in Git is basically one version of a project. Every commit has a commit hash. A commit hash is a unique 40 character hash, which consists of 40 letters and numbers and acts like a name for a commit, or a way to refer to it.

[ NOTE ]

We only really need the first seven characters of a commit hash in order to refer to a commit. For example, if we have a commit has that is 51dc6ecb327578cca508abba4a56e8c18f3835e1 then to refer to this commit we could also just use 51dc6ec.

Every commit represents a snapshot of a project, in other words, a standalone version of a project that contains a reference to all the files that are part of that commit.

Now that we have an idea of what commits are, let’s introduce our final area, the commit history.

## **Introducing the commit history**

The commit history is basically where we can think of our commits existing. We can think of the commit history as being represented by the objects directory in our .git directory. To understand the commit history in depth we would have to dive into the internals of Git but that is a complex topic that we don’t need to know about when learning the basics of how to use Git. For our purposes, all we need to know is that every time we make a commit, it is saved in the commit history.

Let’s go onto Follow Along 2-3 and identify the commit history in our rainbow project directory.

| [ FOLLOW ALONG 2-3 ] |
| --- |
| Inside your rainbow project directory, look inside the .git directory and find the objects directory. |

In Visualize It 2-6 we add a representation for the commit history to our visual model inside the repository.

Now that we have a visual model of the most important areas when working with Git, let’s add the first file to our project. But before we do that, we need a text editor.

**[ VISUALIZE IT 2-6 ]**

A picture containing screenshot, diagram, line, rectangle

Description automatically generated

Adding a representation for the commit history in our diagrammatic mental model

## **Preparing our text editor**

To add a file to our project directory we will use a text editor. A text editor is a program that allows a user to edit text. There are two categories of text editors, graphical text editors and command line text editors. For now we will focus on graphical text editors (referred to simply as text editors henceforth). We will learn more about command line text editors in Chapter 8.

You will need a (graphical) text editor to follow along with the rest of the exercises in this book. If you don’t already have one then in the upcoming Follow Along you can find one and download it. You may consult online resources to find one that meets your requirements. As of writing this book, a couple popular text editors for working on projects using Git are Visual Studio Code[1](https://learning.oreilly.com/library/view/learning-git/9781098133900/lg_ch02.xhtml#footnote-002), Sublime Text[2](https://learning.oreilly.com/library/view/learning-git/9781098133900/lg_ch02.xhtml#footnote-001), and Atom[3](https://learning.oreilly.com/library/view/learning-git/9781098133900/lg_ch02.xhtml#footnote-000). Let’s go onto Follow Along 2-4 and prepare our text editor.

With our text editor ready to go, let’s move on to adding the first file to our rainbow project directory.

| [ FOLLOW ALONG 2-4 ] |
| --- |
| Choose your preferred text editor. If you don’t already have one, then download a text editor. |
| Open the rainbow project directory in a text editor window. |

## **Adding a file to our project directory**

As we mentioned earlier in the book, throughout this learning journey we will work on a project listing the colors of the rainbow and also some colors that are not part of the rainbow. Every time we add a color, we will make a commit to keep track of how our project progresses.

First, we will start by listing the colors of the rainbow. To do so we will create a file called rainbowcolors.txt and in that file we will state that “Red is the first color of the rainbow.” Let’s go ahead to Follow Along 2-5 and make this happen.

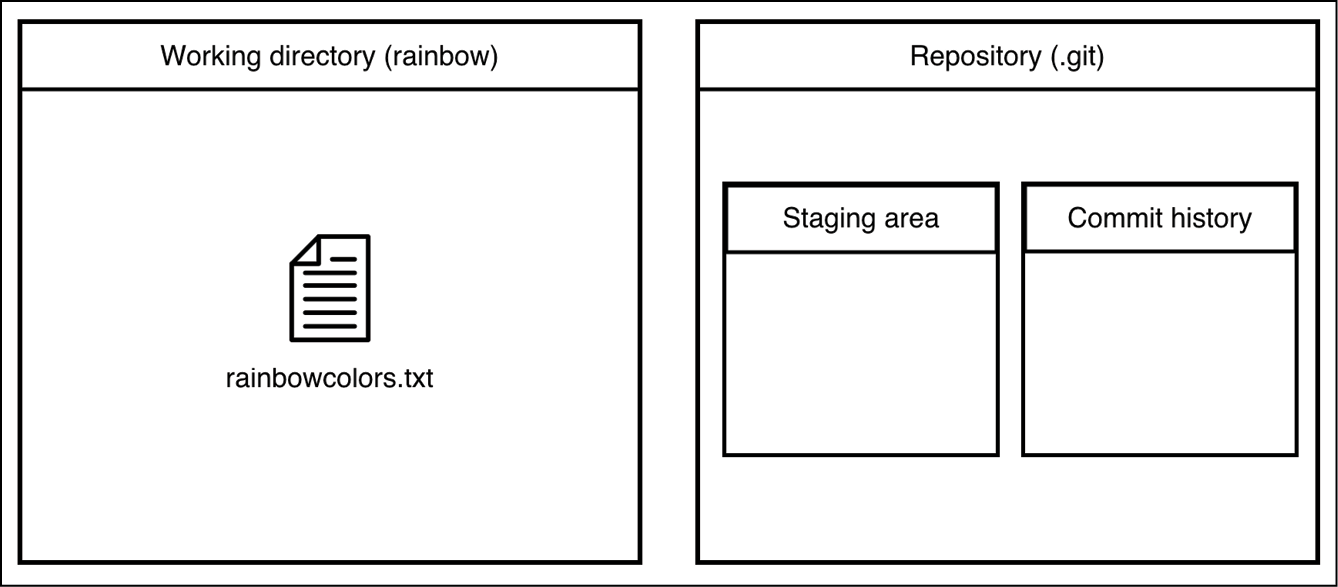
| [ FOLLOW ALONG 2-5 ] |
| --- |
| Create a file called rainbowcolors.txt inside your rainbow project directory. You may use your text editor to do this or create the file directory in your file system. |
| Open the rainbowcolors.txt file in your text editor and on line 1, add “Red is the first color of the rainbow.” and save the file. |

What to notice:

* The rainbowcolors.txt file is in our working directory, we can see it is inside our rainbow project directory.

Even though the rainbowcolors.txt file is in our working directory, it is not yet in our repository. It has not been added to the staging area and it has not yet been included in a commit in the commit history. We illustrate this in Visualize It 2-6 and Visualize It 2-7.

**[ VISUALIZE IT 2-6 ]**



We added a file to our working directory.

**[ VISUALIZE IT 2-7 ]**

A picture containing screenshot, text, diagram, rectangle

Description automatically generated

Our rainbow project directory, before and after we add our rainbowcolors.txt file.

Since the rainbowcolors.txt file is not yet in our repository, it is an untracked file.

An untracked file is a file in the working directory that Git is not version controlling. It has never been added to the staging area and it has never been included in a commit therefore it is not part of the repository.

On the other hand, once we add a file to the staging area and Git becomes aware of it, it becomes a tracked file.

A tracked file is a file that is version controlled or in other words that Git knows about.

Every new file in a project version controlled by Git needs to be explicitly added to the staging area and then included in a commit in order to become a tracked file. We will carry out these steps in Chapter 3.

[ star SUPER IMPORTANT NOTE star ]

Listing the colors of the rainbow in one file is not a very realistic example.

When you are working with your Git version-controlled project you will most probably edit multiple files instead of just one file.

If you’re a developer, you may be writing code.

If you’re a technical writer, you may be writing documentation.

If you’re editing files for any other purpose, again you will most probably edit multiple files in many different ways.

We aim to keep the edits we make to files throughout this learning journey simple so that we can focus on how Git works instead of the contents of the files we are editing. However, as a learner, you should keep in mind that the changes you will be making to files in real projects will look very different.