Let’s create our first Git repository. First we need to create a directory:

```
cd
mkdir my-first-repo
cd my-first-repo

```

“Repo” in this case is just shorthand for “repository.” To start tracking files with Git in a directory enter git init into the command line:

```
git init

## Initialized empty Git repository in /Users/sean/my-first-repo/.git/
```

You’ve just created your first repository! Now let’s create a file and start tracking it.

```
echo "Welcome to My First Repo" > readme.txt
```

Now that we’ve created a file in this Git repository, let’s use git status to see what’s going on in this repository. We’ll be using git status continuously throughout this chapter in order to get information about the status of this Git repository.

```
git status

## On branch master
##
## Initial commit
##
## Untracked files:
##   (use "git add <file>..." to include in what will be committed)
##
##  readme.txt
##
## nothing added to commit but untracked files present (use "git add" to track)
```

As you can see readme.txt is listed as an untracked file. In order to let Git know that you want to track this file we need to use git add with the name of the file that we want to track. Let’s start tracking readme.txt:

```
git add readme.txt

```

Git now knows to track any changes to readme.txt. Let's see how the status of the repository has changed:

```
git status

## On branch master
##
## Initial commit
##
## Changes to be committed:
##   (use "git rm --cached <file>..." to unstage)
##
##  new file:   readme.txt
##
```

Git is now tracking readme.txt, or in Git-specific language readme.txt is now staged. Between the parentheses in the message above you can see that git status is giving us a tip about how to unstage (or un-track) this file, which we could do with git rm --cached readme.txt. Let's unstage this file just to see what happens:

```
git rm --cached readme.txt

## rm 'readme.txt'

git status

## On branch master
##
## Initial commit
##
## Untracked files:
##   (use "git add <file>..." to include in what will be committed)
##
##  readme.txt
##
```

Our repository is right back to the way it started with readme.txt as an unstaged file. Let's start tracking readme.txt again so we can move on to cooler Git features.

```
git add readme.txt
```

Now that Git is tracking readme.txt we need to create a milestone to indicate the changes that we made to readme.txt. In this case, the changes that we made were creating the file in the first place! This milestone is called a commit in Git. A commit logs the content of all of the currently staged files. Right now we only have readme.txt staged so let's commit the creation of this file. When making a Git commit, we need to write a commit message which is specified after the -m flag. The message should briefly describe what changes you've made since the last commit.

```
git commit -m "added readme.txt"

## [master (root-commit) 73e53ca] added readme.txt
##  1 file changed, 1 insertion(+)
##  create mode 100644 readme.txt
```

The message above confirms that the commit succeeded and it summarizes the changes that took place since the last commit. As you can see in the message we only changed one file, and we only changed one line in that file. Let’s run git status again to see the state of our repository after we’ve made the first commit:

```
git status

## On branch master
## nothing to commit, working tree clean
```

All of the changes to the files in this repository have been committed! Let’s add a few more files to this repository and commit them.

```
touch file1.txt
touch fil2.txt
ls

## file1.txt
## fil2.txt
## readme.txt
```

While we’re at it let’s also add a new line of text to readme.txt:

```
echo "Learning Git is going well so far." >> readme.txt

```

Now that we’ve added two more files and we’ve made changes to one file let’s take a look at the state of this repository.

```
git status

## On branch master
## Changes not staged for commit:
##   (use "git add <file>..." to update what will be committed)
##   (use "git checkout -- <file>..." to discard changes in working directory)
##
##  modified:   readme.txt
##
## Untracked files:
##   (use "git add <file>..." to include in what will be committed)
##
##  fil2.txt
##  file1.txt
##
## no changes added to commit (use "git add" and/or "git commit -a")
```

We can see that Git has detected that one file has been modified, and that there are two files in this directory that it is not tracking. Now we need to tell Git to track the changes to these files. We could tell Git to track changes to each file using git add, or since all of the files in this repository are .txt files we could use a wildcard and enter git add *.txt into the console. However if we want to track all of the changes to all of the files in our directory we should use the command git add -A.

```
git add -A
git status

## On branch master
## Changes to be committed:
##   (use "git reset HEAD <file>..." to unstage)
##
##  new file:   fil2.txt
##  new file:   file1.txt
##  modified:   readme.txt
##
```

Now the changes to all of the files in this repository are being tracked. Finally let’s commit these changes:

```
git commit -m "added two files"

## [master 53a1983] added two files
##  3 files changed, 1 insertion(+)
##  create mode 100644 fil2.txt
##  create mode 100644 file1.txt
```

Darn it, now looking at this commit summary I realize that I have a typo in one of the names of my files! Thankfully we can undo the most recent commit with the command git reset --soft HEAD~:

```
git reset --soft HEAD~
git status

## On branch master
## Changes to be committed:
##   (use "git reset HEAD <file>..." to unstage)
##
##  new file:   fil2.txt
##  new file:   file1.txt
##  modified:   readme.txt
##
```

This repo is now in that exact same state it was right before we made the commit. Now we can rename fil2.txt to file2.txt, then let’s look at the status of the repository again.

```
mv fil2.txt file2.txt
git status

## On branch master
## Changes to be committed:
##   (use "git reset HEAD <file>..." to unstage)
##
##  new file:   fil2.txt
##  new file:   file1.txt
##  modified:   readme.txt
##
## Changes not staged for commit:
##   (use "git add/rm <file>..." to update what will be committed)
##   (use "git checkout -- <file>..." to discard changes in working directory)
##
##  deleted:    fil2.txt
##
## Untracked files:
##   (use "git add <file>..." to include in what will be committed)
##
##  file2.txt
##
```

We previously told Git to track fil2.txt, and we can see that Git acknowledges that the file has been deleted. We can bring Git up to speed with what files it should be tracking with git add -A:

```
git add -A
git status

## On branch master
## Changes to be committed:
##   (use "git reset HEAD <file>..." to unstage)
##
##  new file:   file1.txt
##  new file:   file2.txt
##  modified:   readme.txt
##

```

Finally we got the file names right! Now let's make the correct commit:

```
git commit -m "added two files"

## [master 12bb9f5] added two files
##  3 files changed, 1 insertion(+)
##  create mode 100644 file1.txt
##  create mode 100644 file2.txt

```

That looks much better.

Summary

- Git tracks changes to plain text files (code files and text documents).
- A directory where changes to files are tracked by Git is called a Git repository.
- Change your working directory, then run git init to start a repository.
- You can track changes to a file using git add [names of files].

- You can create a milestone about the state of your files using git commit -m "message about changes since the last commit".
- To examine the state of files in your repository use git status.

Mark as completed