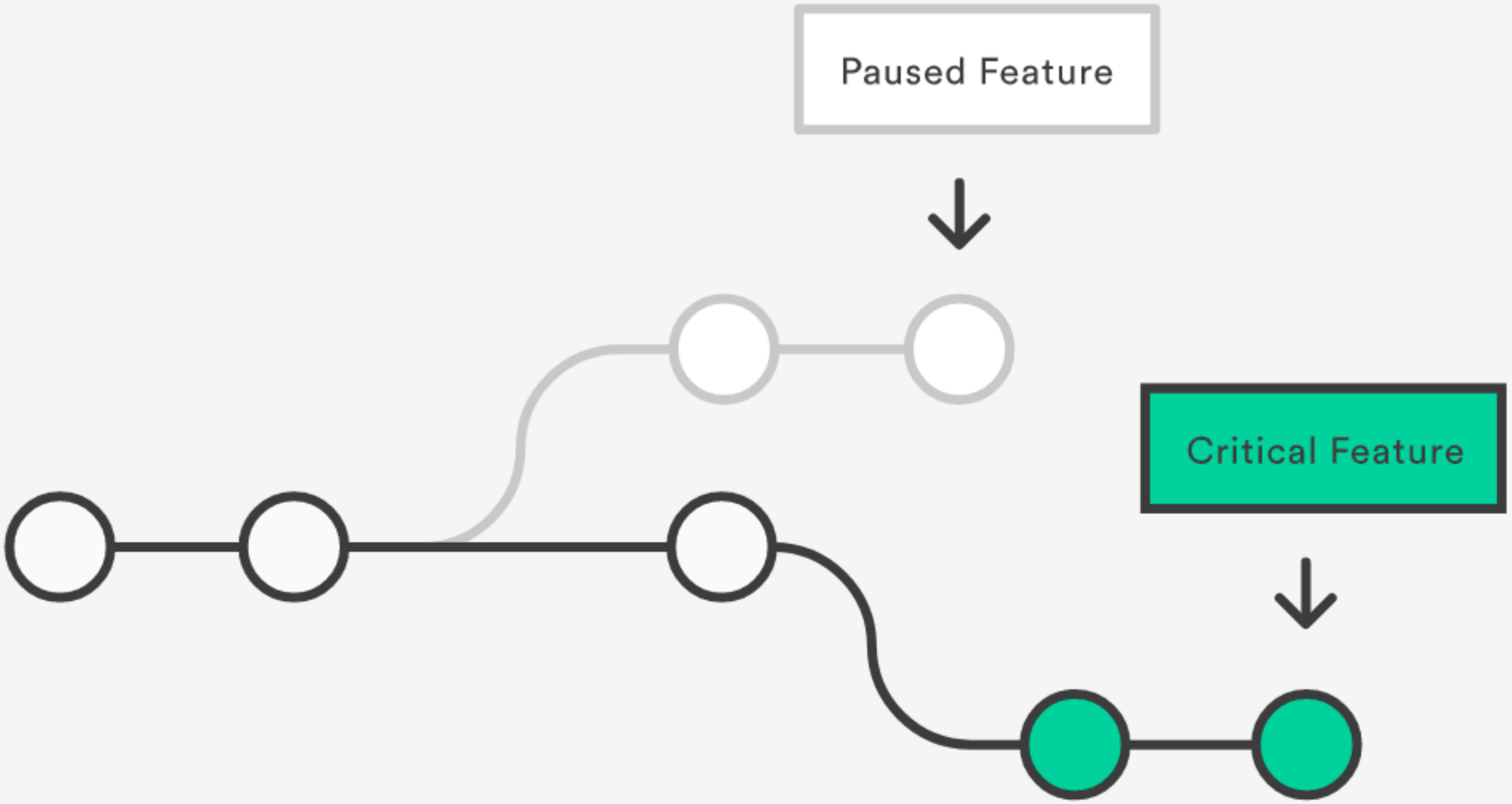
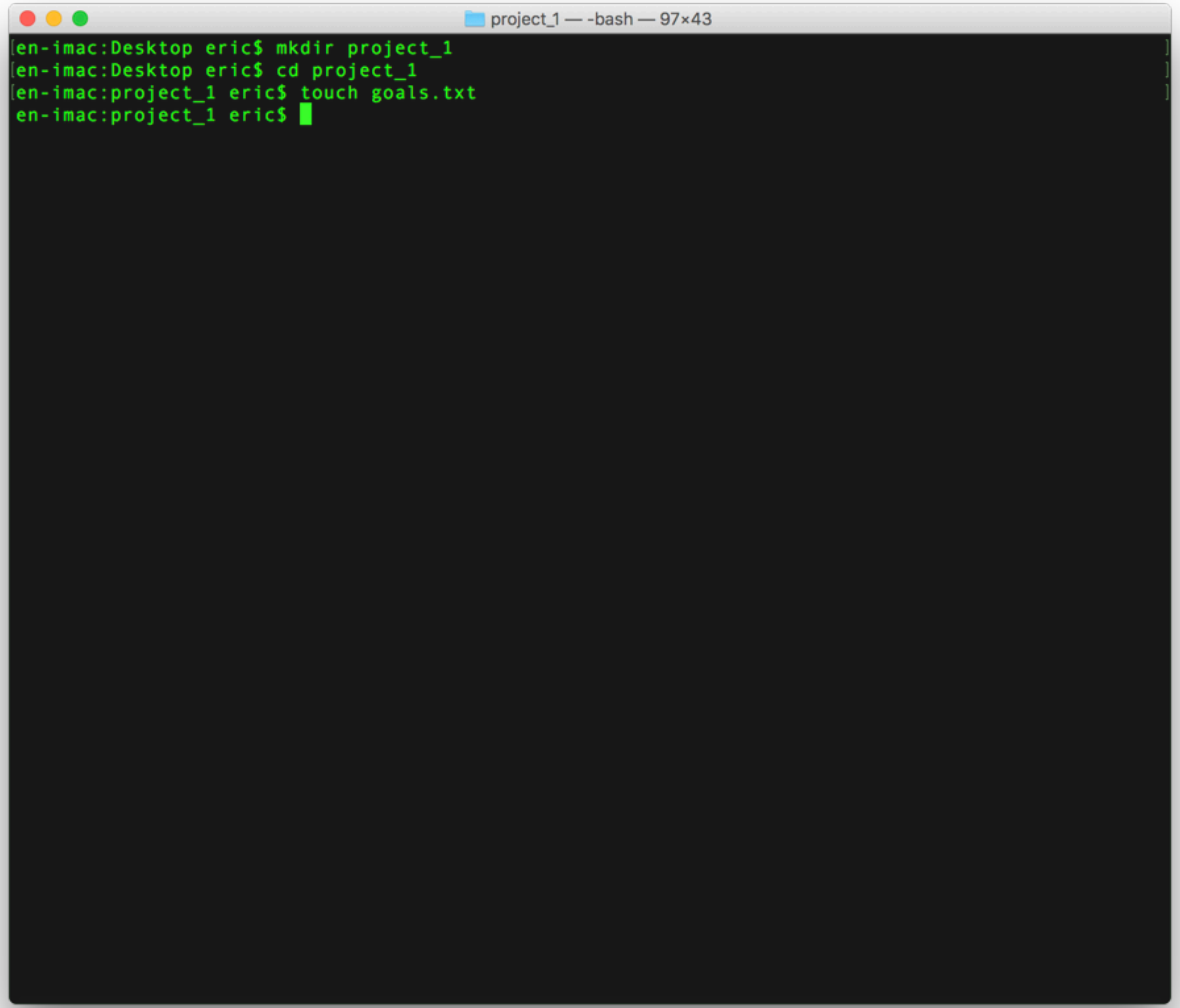
GIT

Git is a version control system

Version control systems are a category of software tools that help a software team manage changes to source code over time.

Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.

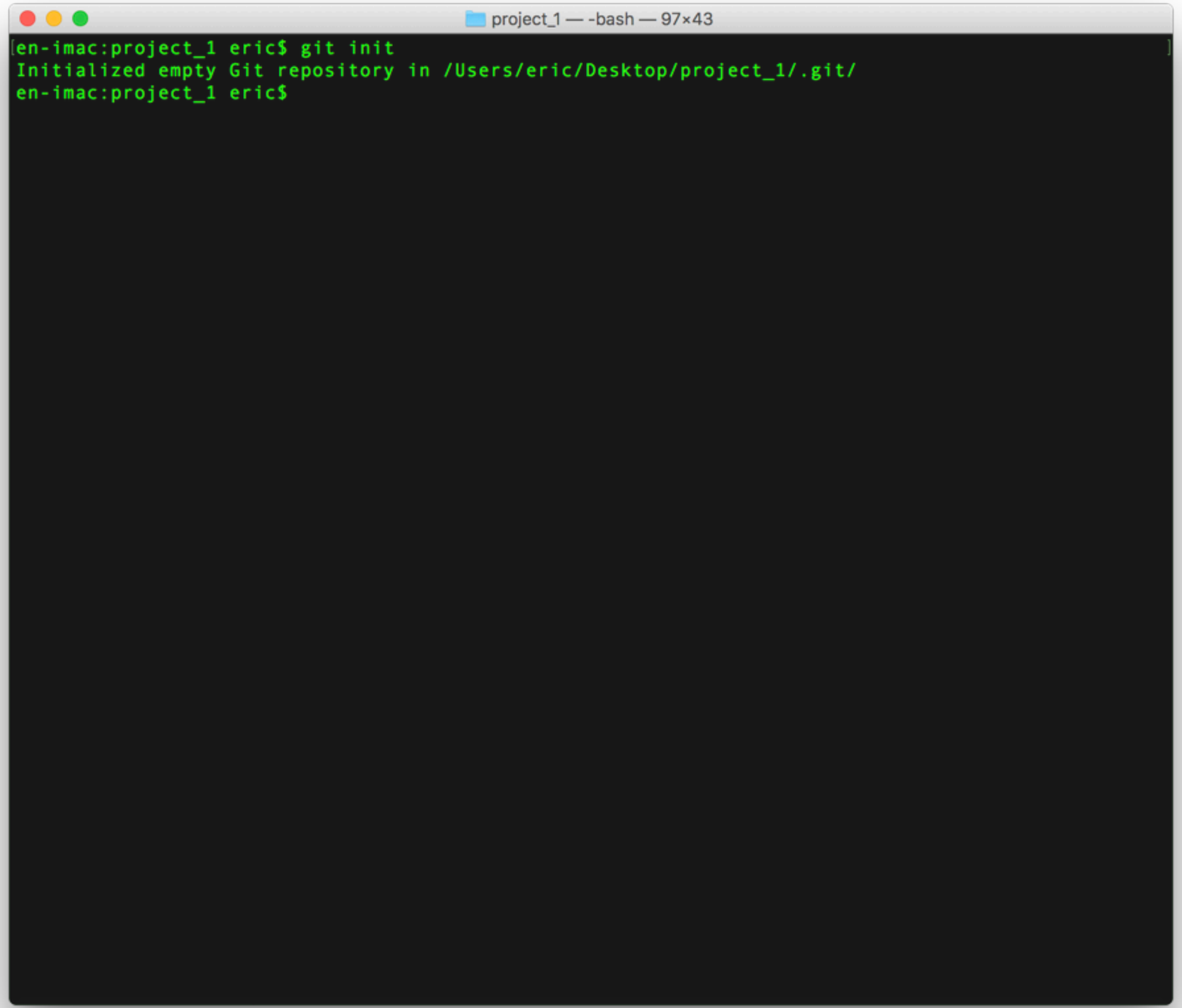

Paused Feature
Critical Feature


project_1 — -bash — 97×43

```
en-imac:Desktop eric$ mkdir project_1
en-imac:Desktop eric$ cd project_1
en-imac:project_1 eric$ touch goals.txt
en-imac:project_1 eric$ 
```

Let’s try *initializing a git repository*.

First make a directory for your project and add a text file that you want to track with git.


project_1 — -bash — 97×43

```
en-imac:project_1 eric$ git init
Initialized empty Git repository in /Users/eric/Desktop/project_1/.git/
en-imac:project_1 eric$
```

Now that we have started working on project 1, let's turn the project_1 directory into a Git project. We do this with `git init`.

The word "init" stands for *initialize*.

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

1

Working Directory

```
Make changes to
files:

+ additions
- deletions
  modifications
```

2

Staging Area

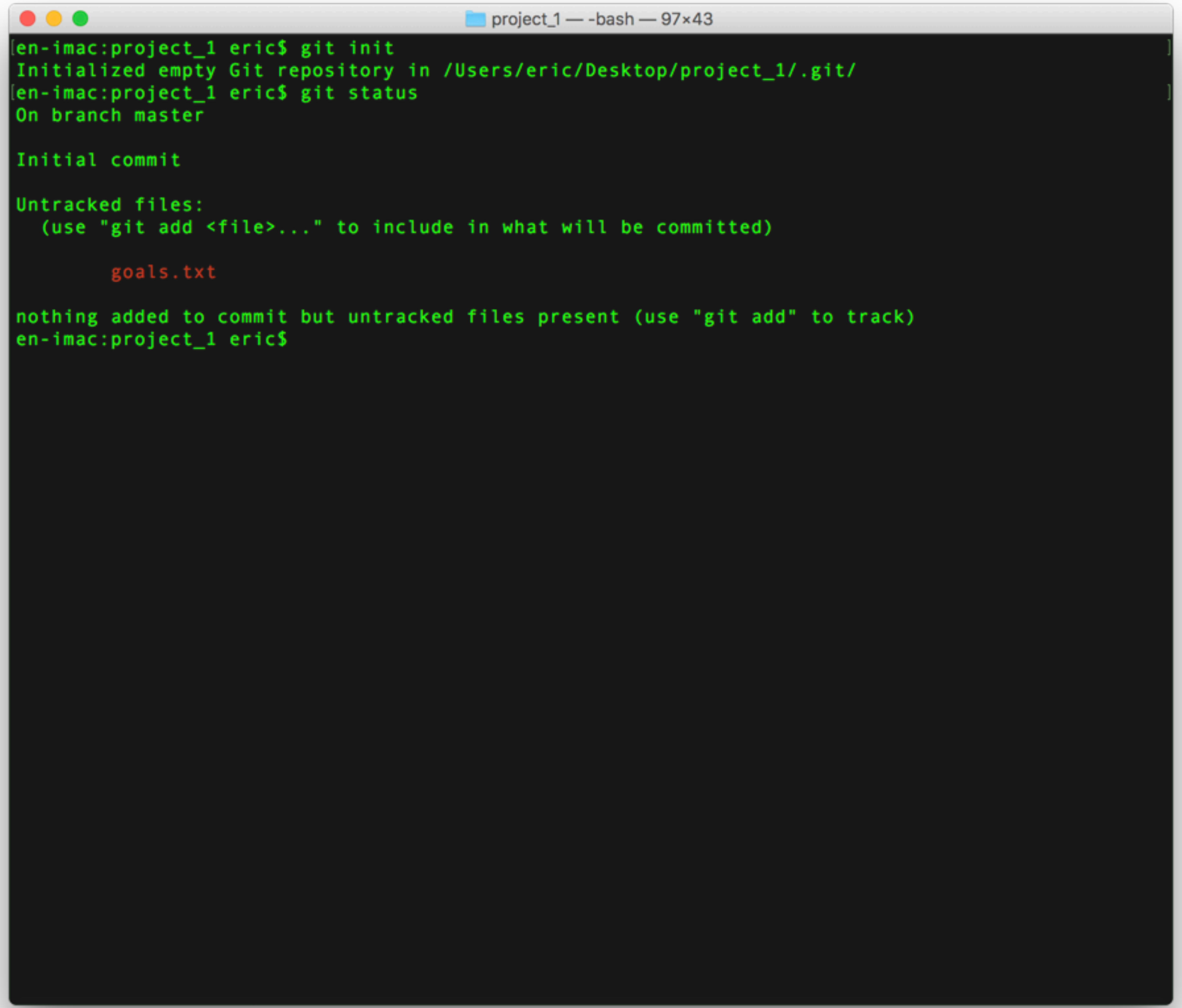
```
Bring changes into
the staging area
```

3

Repository

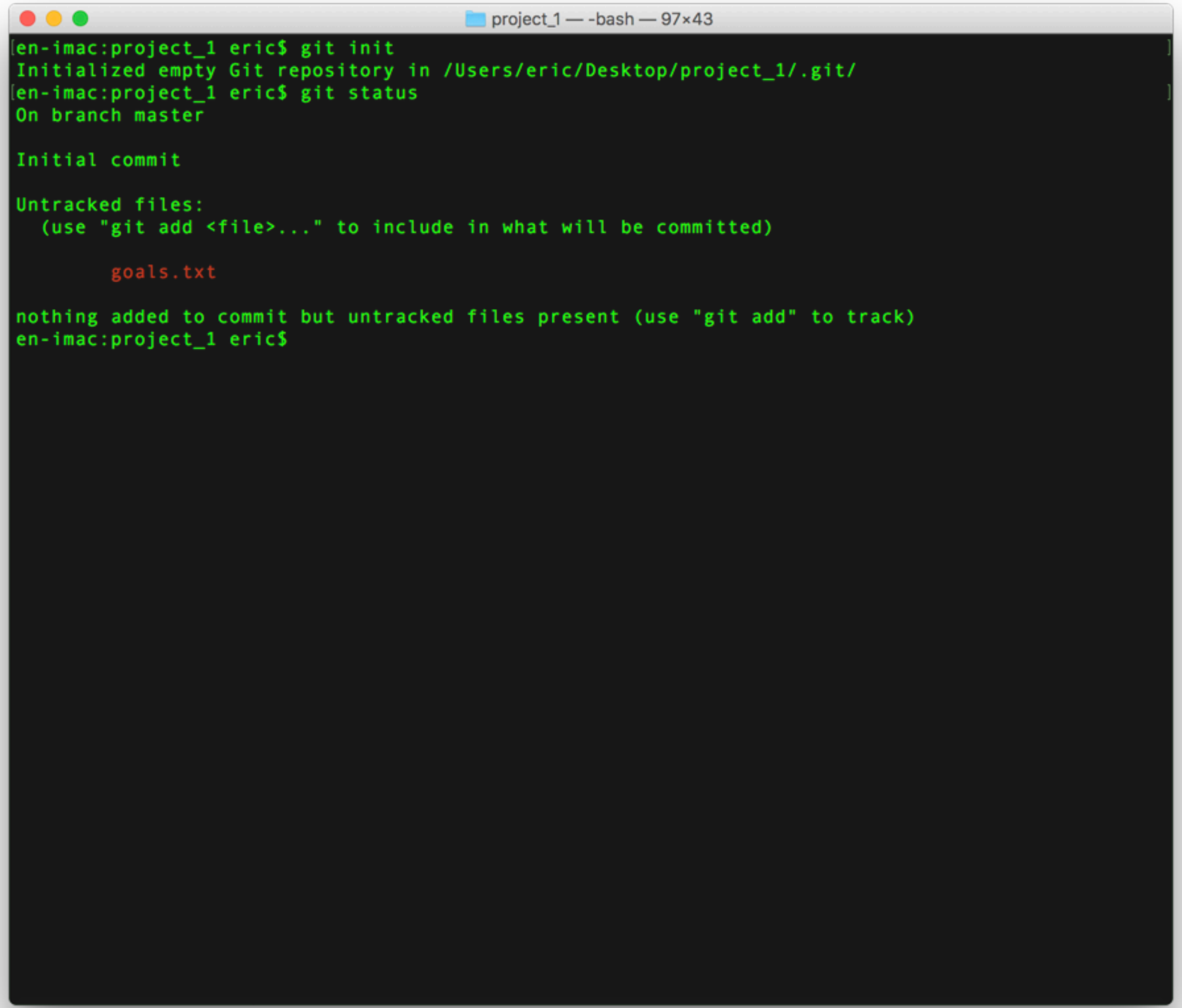
```
Save changes to the
repository as a
‘commit’
```

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*.



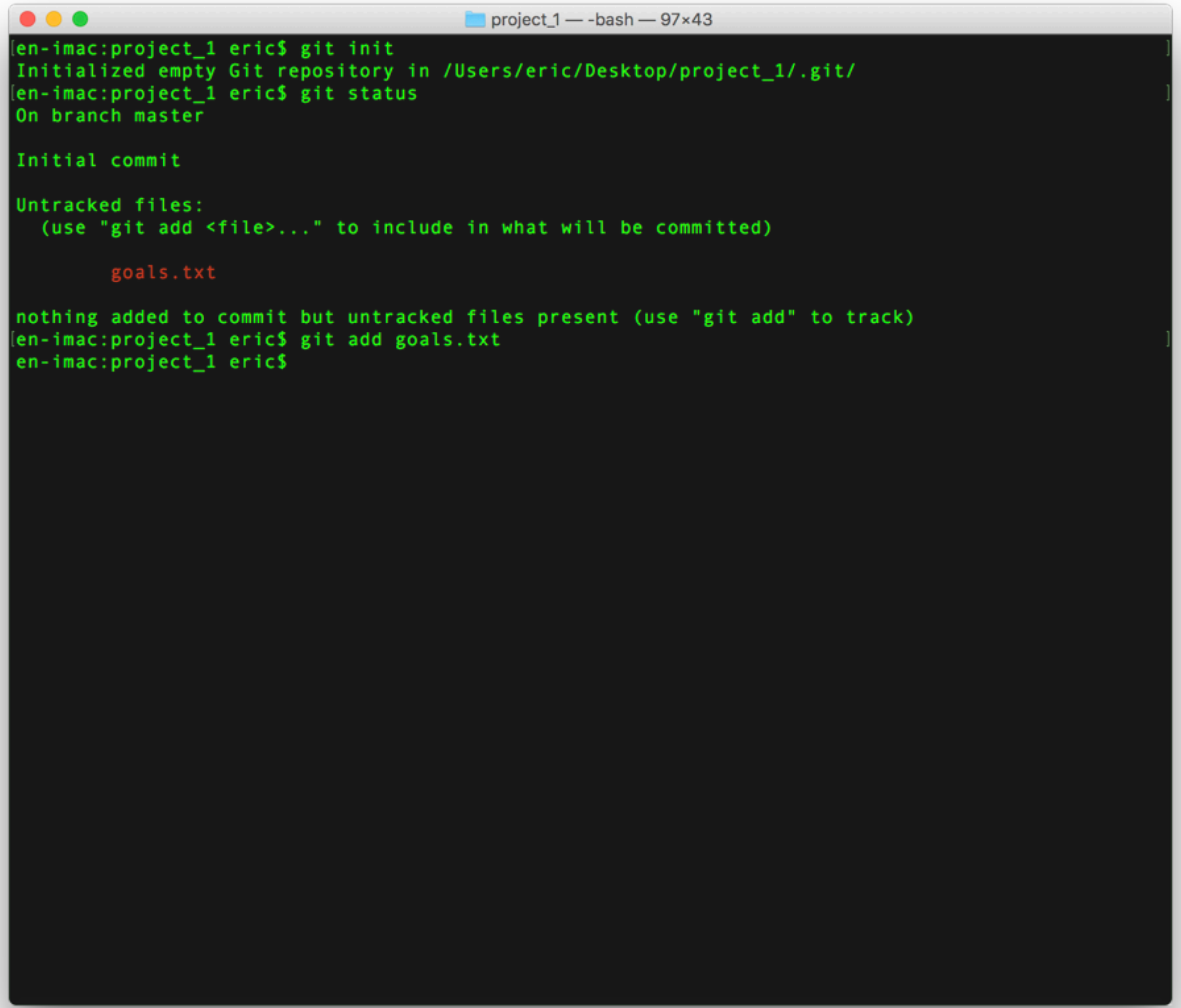
```
en-imac:project_1 eric$ git init
Initialized empty Git repository in /Users/eric/Desktop/project_1/.git/
en-imac:project_1 eric$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        goals.txt

nothing added to commit but untracked files present (use "git add" to track)
en-imac:project_1 eric$
```

As you work on the project, you will be changing the contents of the working directory. You can check the status of those changes with `git status`



```
en-imac:project_1 eric$ git init
Initialized empty Git repository in /Users/eric/Desktop/project_1/.git/
en-imac:project_1 eric$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        goals.txt

nothing added to commit but untracked files present (use "git add" to track)
en-imac:project_1 eric$
```

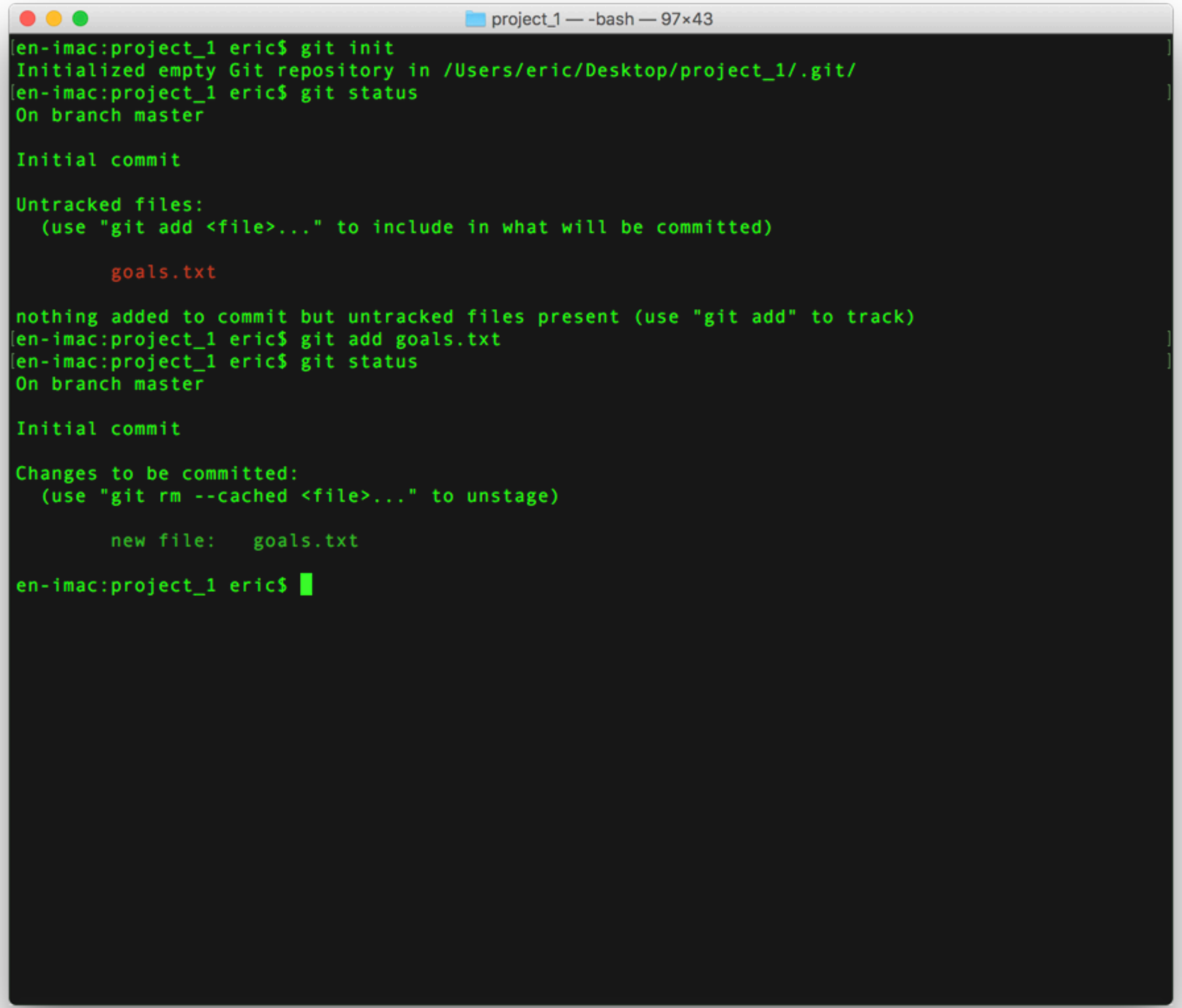
In the output, notice the file in red under **untracked files**. Untracked means that Git sees the file but has not started tracking changes yet.



```
en-imac:project_1 eric$ git init
Initialized empty Git repository in /Users/eric/Desktop/project_1/.git/
en-imac:project_1 eric$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        goals.txt

nothing added to commit but untracked files present (use "git add" to track)
en-imac:project_1 eric$ git add goals.txt
en-imac:project_1 eric$
```

In order for Git to start tracking goals.txt, the file needs to be added to the staging area.

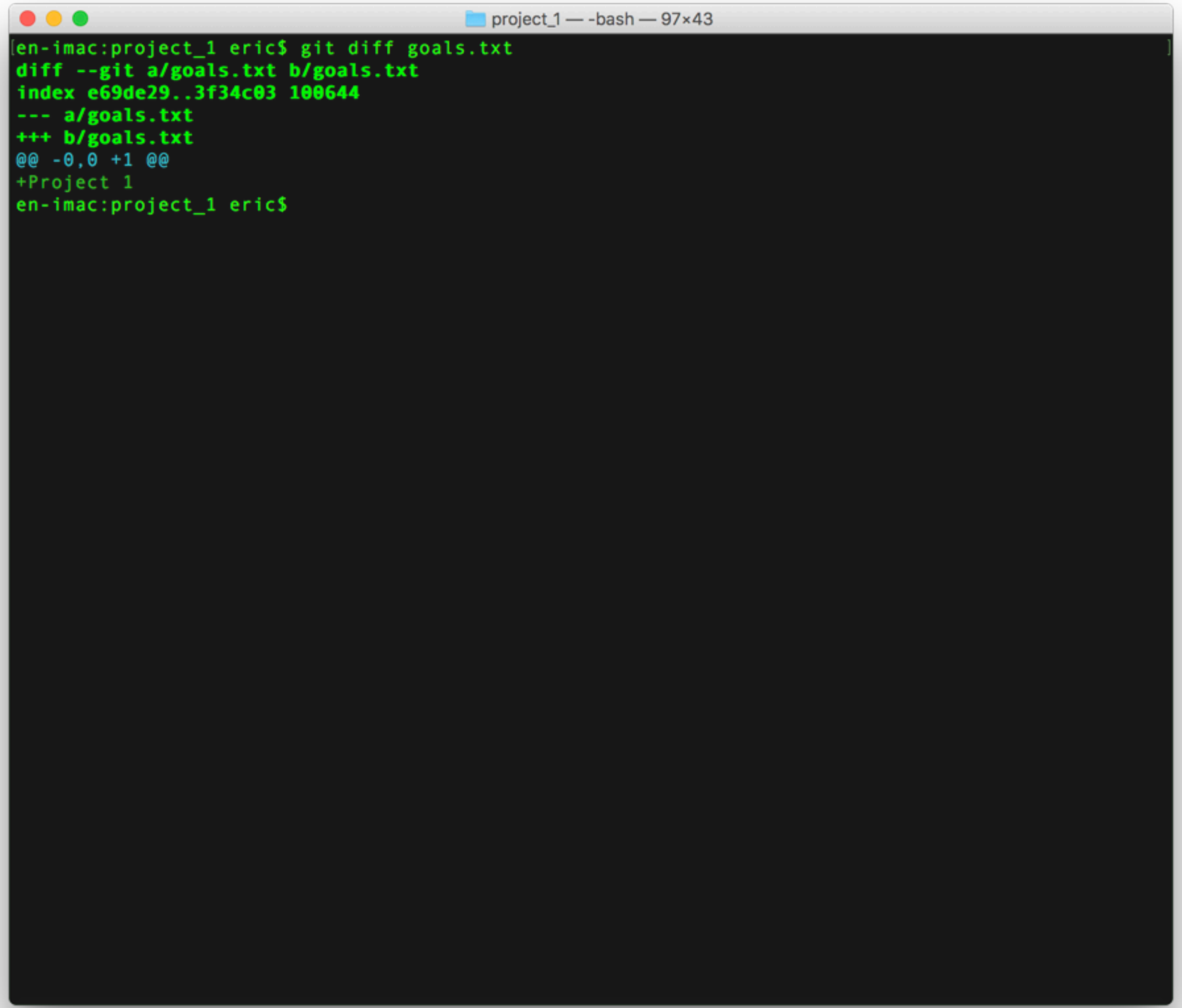
We can add a file to the staging area with `git add filename`. The word `filename` here refers to the name of the file you are editing, such as goals.txt.



```
en-imac:project_1 eric$ git init
Initialized empty Git repository in /Users/eric/Desktop/project_1/.git/
en-imac:project_1 eric$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        goals.txt

nothing added to commit but untracked files present (use "git add" to track)
en-imac:project_1 eric$ git add goals.txt
en-imac:project_1 eric$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   goals.txt

en-imac:project_1 eric$ 
```

Check the status of the project in Git.

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



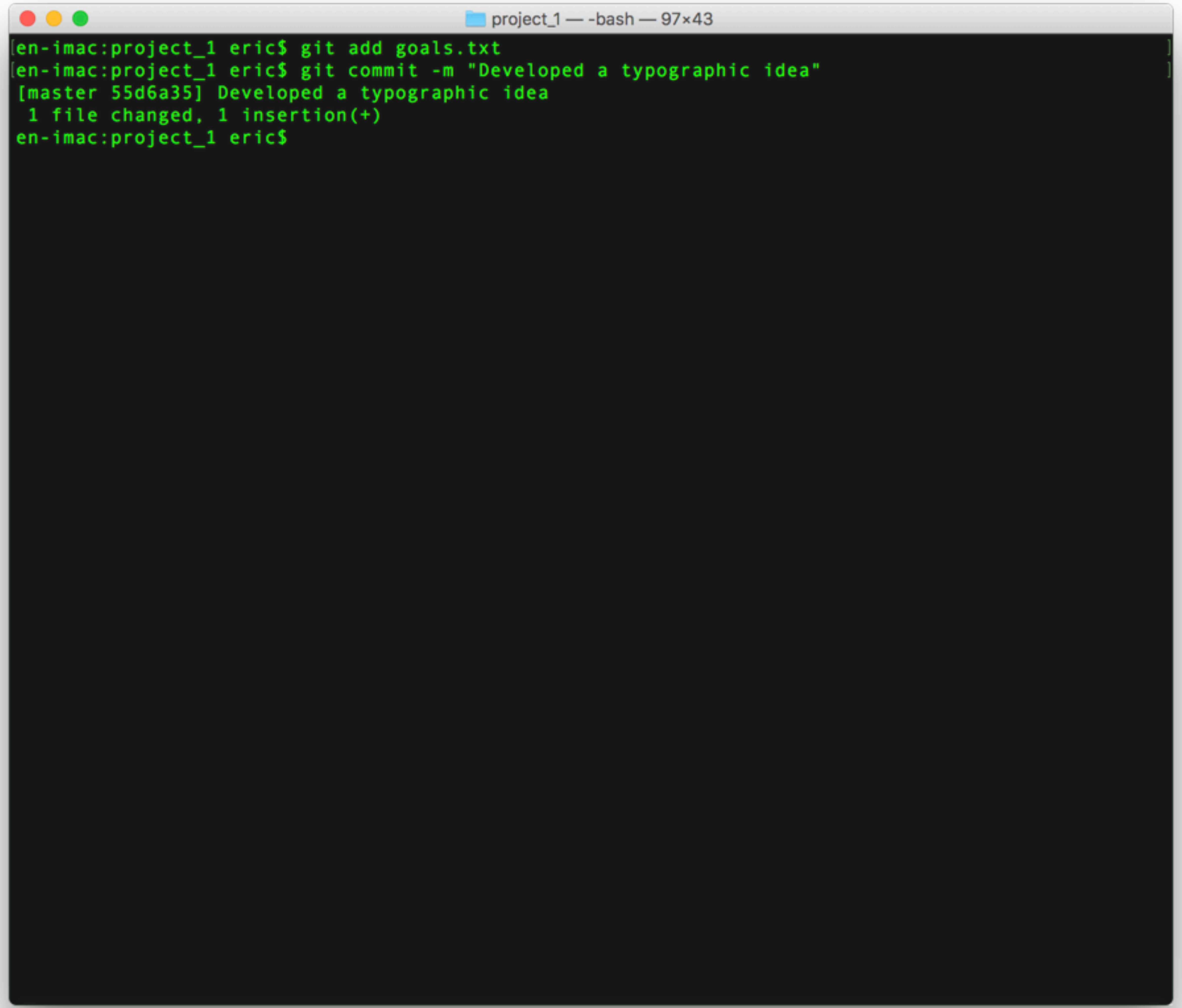
```
en-imac:project_1 eric$ git diff goals.txt
diff --git a/goals.txt b/goals.txt
index e69de29..3f34c03 100644
--- a/goals.txt
+++ b/goals.txt
@@ -0,0 +1 @@
+Project 1
en-imac:project_1 eric$
```

Imagine that we type another line in goals.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 goals.txt the command would be `git diff goals.txt`

Each time you make a set of changes you will need to use `git add filename` to add the files to the staging area.

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

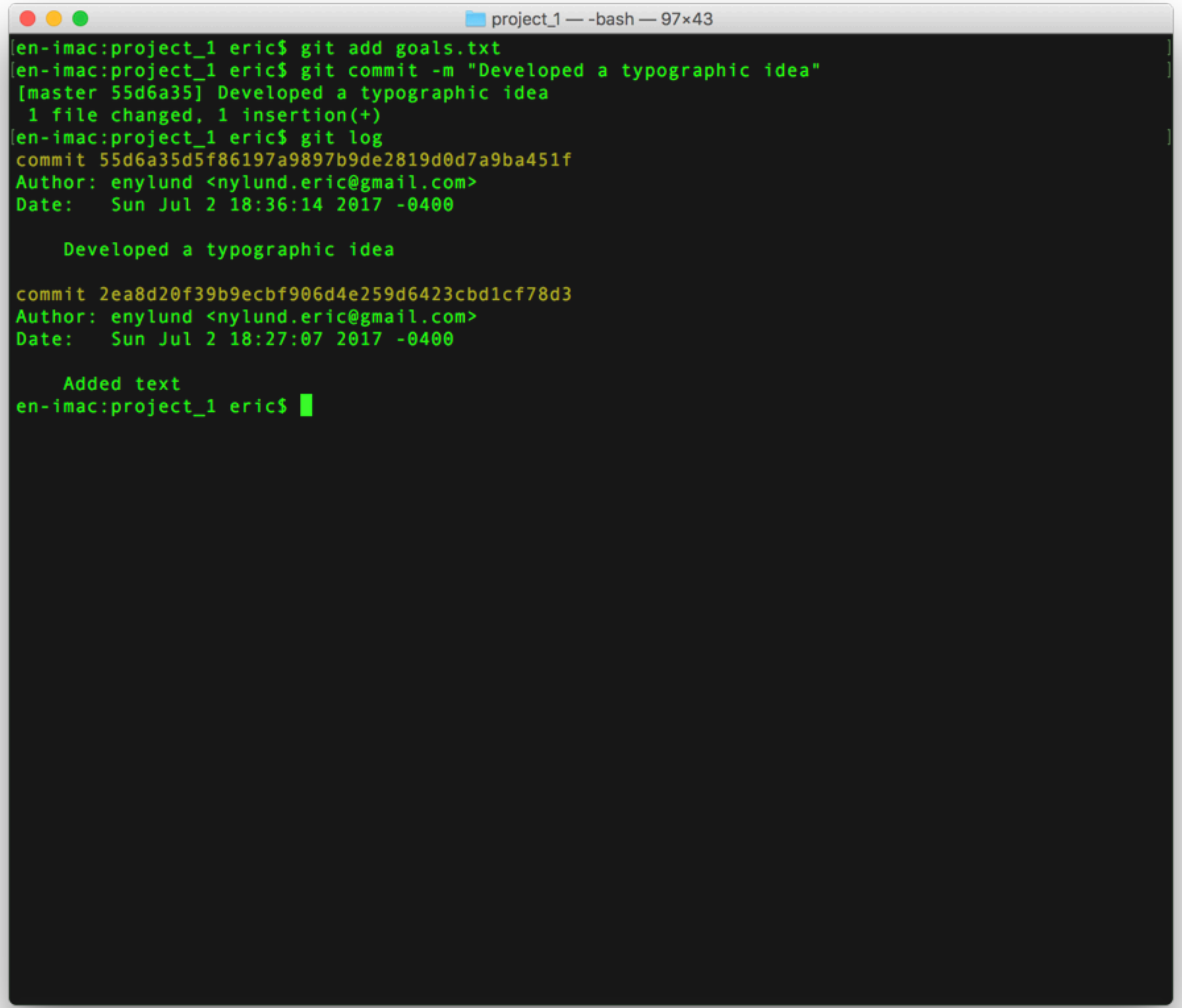


```
en-imac:project_1 eric$ git add goals.txt
en-imac:project_1 eric$ git commit -m "Developed a typographic idea"
[master 55d6a35] Developed a typographic idea
 1 file changed, 1 insertion(+)
en-imac:project_1 eric$
```

`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 “Developed a typographic idea“`

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`



```
en-imac:project_1 eric$ git add goals.txt
en-imac:project_1 eric$ git commit -m "Developed a typographic idea"
[master 55d6a35] Developed a typographic idea
 1 file changed, 1 insertion(+)
en-imac:project_1 eric$ git log
commit 55d6a35d5f86197a9897b9de2819d0d7a9ba451f
Author: enylund <nylund.eric@gmail.com>
Date:   Sun Jul 2 18:36:14 2017 -0400

    Developed a typographic idea

commit 2ea8d20f39b9ecbf906d4e259d6423cbd1cf78d3
Author: enylund <nylund.eric@gmail.com>
Date:   Sun Jul 2 18:27:07 2017 -0400

    Added text
en-imac:project_1 eric$ 
```

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

```
git init

git status

git add

git diff

git commit

git log
```

git init	creates a new Git repository
git status	inspects 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