Git for Beginners

By Hannah Ellis

Content

Who is this book for?

What is Git?

The Structure of Git

When working with git in your day to day it's useful to know about the way git is structured. There are four main working areas that you'll be using when working with git. These four areas are

- The Working Directory
- The Staging Area (Also know as the index)
- The Local Repository
- The Remote Repository

The further down the above list you go the less control you have over changes to indivdual files and the more control git has. Lets look at each of these areas together now.

The Working Directory

The working directory is where you make all your changes. It is the directory which is under version control using git. You have the most control over this area and can make changes using editors without ever typing a single git command. While there are git commands that can make changes in this area, git commands are not the only way unlike the next area.

The Staging Area (Also know as the index)

The Staging Area or Index is where you stage your changes to your working directory ready to be put under git's full control. This is the area between you and your local repository. The main purpose of this area is to help you organise how you label the changes you wish to put

under git's full control. You will also see this area come into play a lot when you have conflicts between changes. You can only make changes here using git commands, but you have still have control over what goes under git's full control.

The Local Repository

The Local Repository is what allows git to be distributed. You have your own local copy of the reposity that you can control with git commands. Once you have put changes into the local repository you cannot edit them easily. It is possible, but you can only really edit blocks of changes that were made in the staging area prevously.

Also the Local Repository can be seen a little bit like a staging area for the remote repository.

The Remote Repository

The remote repository is the area that everyone on your team and the outside world can see your changes. You can think of the remote repository as the perfect copy of your code (The good enough to ship version). As such you tend to have less control over this area. It is possible to have more than one remote repository in your project, but most likely you'll only ever work with one.

Working locally

It is possible to work with git without having a remote repository at all. For this first section that is exactly what we will do. The reason for this is so we can just think about the interaction between the first three areas, and since this is what you will most likely be doing mostly, it's a good place to start and for you to get some practice.

Initilising your Local Repository

The first thing you will want to do is identify the directory you want to put under version control. This directory is your working directory as mentioned before as one of the working areas in git. This directory can be empty or can already contain files. In your terminal either create or navigate to the directory you wish to put under version control. You can do this by using the `cd` and `mkdir` commands if using a bash shell.

Once your terminal is in your directory, you can use the following command to create your local repository.

```
git init
```

That is it. You now have a local reposity you can work with. This repository is currently empty and knows nothing about the files you might already have in your working directory. Additionaly your staging are is also empty.

Before we start looking at adding changes to the staging area, let's have a look at what things look like currently to git. It is often useful to be able to see what git knows about your working directory and what state things are in generally. The command that can tell you about the status of git is simpliy

```
git status
```

As an example here is what `git status` tells me after initialising an empty working directory

```
working % git status
On branch main

No commits yet

nothing to commit (create/copy files and use
"git add" to track)
```

Just to disect this a little bit. The first line tells you which branch you are on. Don't worry about branches for now, we will cover those in a later section. The next line tells you about commits, this we will cover those shortly. The last line tells you that there's no changes in your working directory.

The usage of git status can go a little bit further too. You can provide a file or a directory after the command if you only want to know about the status of that one file or directory, the command looks like this

```
git status <file or directory>
```

where you replace `<file or directory>` with whichever file or directory you wish.

Making changes to your working directory

If you are working from an empty directory then you won't have any changes yet. If you already do because you didn't start with an empty directory, then feel free to jump to the next section.

If you do not have any changes yet then with your favorite editor make a new file, let's call it "my_new_file.txt" and inside it we will add the text

> This is my first file

Once you have made this change and saved it. You will now have a change in your working directory. Let's see what git thinks of it by running git status once again.

```
working % git status
On branch main

No commits yet

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

nothing added to commit but untracked files
present (use "git add" to track)
```

Now that we have some changes, lets continue to see what we can do with them.

Checking the differences

Now that you have a change in your working directory, you can get git to show you the difference between what is in your working directory and what is in the staging area (more about the staging area later). To show the difference you can use

```
git diff
```

As with many git commands you can include a list of files or directories at the end of this command to just see those differences. Right now because we're just adding a new file, we won't see any differences being shown. We will revisit this command in a later section after we've added some changes to the staging area.

Removing a change from your working directory

Adding files to the staging area

Since we don't have any changes yet (maybe you do already if you didn't start with an empty directory, don't worry we'll get onto adding these soon.), we should make some. With and editor make a new file, let's call it "my_new_file.txt" and inside it we will add the text

> This is my first file

Once you have made this change and saved it. You will now have a change in your working directory. Let's see what git thinks of it by running git status once again.

```
working % git status
On branch main

No commits yet

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

```
nothing added to commit but untracked files
present (use "git add" to track)
```

You now see the new file under the untract files list. This means that there are changes in the working directory that have not yet in the staging area in git. So our next step is to add this file. The commnand for this is

```
git add
```

This command takes a filename or a directory or a list of files/directories after it. This tells git to add the listed files or directories to the staging area.

Let's do that now. To add the single file we have so far type `git add my_new_file.txt` into the command line and press enter. Now when we do `git status` we see the following.

```
working % git status
On branch main

No commits yet

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

As you can see the file has moved from *Untracked files* to *Changes to be committed*. This means your changes are now in the staging area.

You can choose to make more changes if you like, but these changes won't end up in the staging area until you add them again. This is because your file is now partly under git's control. You've added it to the staging area and so git remembers the changes made when it was added.

git diff revisited

If you do make more changes (which we will do in this section), then you can see them using `git diff`.

Removing files from the staging area

It's quite possible that you've added files to the staging area you did not mean to. For example you might have added an entire directory but you only meant to add some of the changes inside that directory. Although adding your files puts them more under git's control, you still have some control at this stage. As well as adding changes you can also remove them by using the following command

```
git reset
```

Just like with `git add` you will need to follow this by a list of files or directories you wish to remove from the staging area.

Let's go through an example together.

With and editor make another new file, let's call it "some_changes_i_do_not_want.txt" and inside it we will add the text

> I do not wish to add these changes

Once you have made this change and saved it, and then use `git add` to add this file to the staging area. Hopefully you should be able to do this by yourself now. Feel free to look back if you need a reminder, and to use `git status` if you want to check that it has been added.

Now that you have added this file, we shall now remove it from the staging area by using the following command

```
git reset some_changes_i_do_not_want.txt
```

Now if you do `git status` you will see that this file is no longer in the staging area.

Committing changes

Once you have changes in the staging area you can commit them. Note here that a commit is hard to change, so you should make sure that the changes in the staging area are changes you want to commit.

Showing changes between the last commit and the current staging area

Summary

In this section we have seen

- How to create a local repository using the `git init` command.
- How to add changes to the staging area using the `git add` command.

Creating an account on Github

Creating a repository locally

Setting up a remote on a local repository

Working with a remote repository