---

title: Git FUN!damentals
subtitle: Tracking Files
minutes:

Okay, we have git ready to go, now let's try using it!

Making a repository

Let's start by making a new directory called 'fruits' and navigating to it

```
$ mkdir fruits
$ cd
```

Now, we're going to tell git to start tracking what we're up to

```
$ git init
```

If you look in your directory now, you'll see that there is something called `.git`

```
.     ..    .git
```

Adding files

Cool! but this still isn't going anything for us because this directory is empty. So let's create a file in our new directory

```
$ touch fruit_list.txt
$ nano fruit_list.txt
```

Let's add some fruits:

```
banana
apple
peach
```

Remember `CTRL + O` to write, and `CTRL + X` to exit.

Now let's see what git is up to with `git status`

```
On branch master

Initial commit

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

    fruit_list.txt

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

Here, git is telling you that there are local files that you haven't told git to look at

```
$ git add fruit_list.txt
```

Now, when we run status, git tells us that we've told it to keep track of a new file

```
On branch master

Initial commit

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

    new file:   fruit_list.txt
```

If this was a mistake, we could correct it with `git rm`

Committing changes

Now we're ready to commit this file. Committing changes means making a permanent record of the current state of your repository.

```
$ git commit
```

You'll see something like this:

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#       new file:   fruit_list.txt
#
```

Every commit needs a message to accompany it. It should be as brief as possible while still describing what changes you made.

> Making a whole bunch of changes and committing them all at once is *BAD*. If you make many commit for many small changes, and one of those changes breaks your code, you can *selectively undo* that change and fix the bug. If you make only one commit, you'll have to re-do everything from scratch which is *WET* which is *BAD*

It takes a while to get a hang of this, and it helps to read other people's commits to know what to say

This time, we'll add "fruit_list added" and then type ^x to write and quit

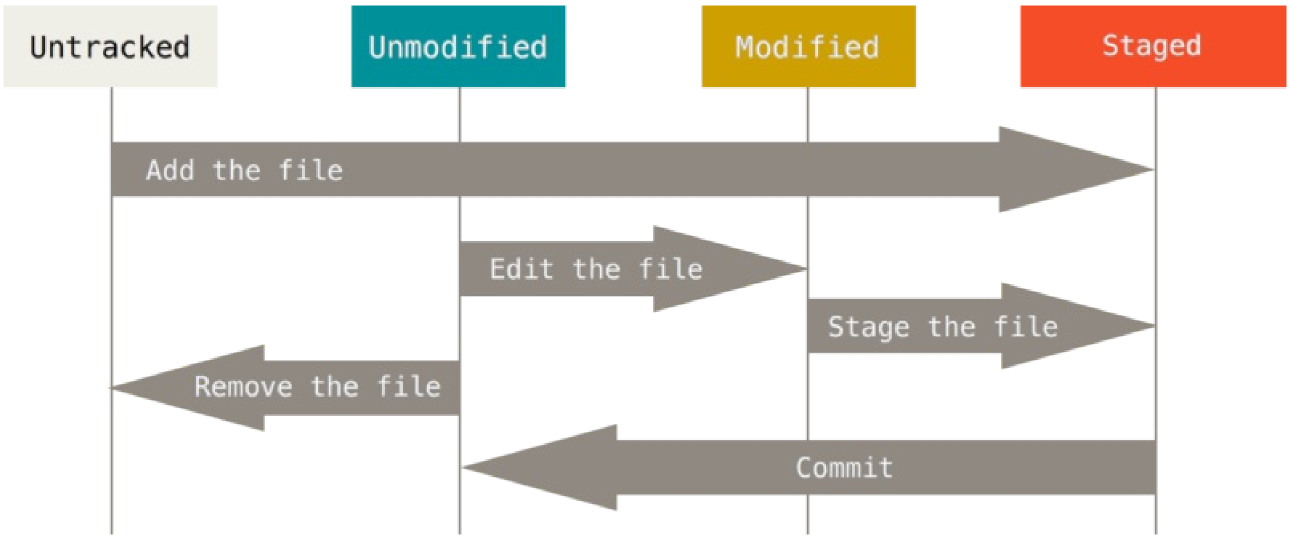
```
[master (root-commit) 07f5ba5] README added
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 fruit_list.txt
```

If you don't want to open nano every time you make a commit, you can use flag m

```
$ git commit -m "fruit_list added"
```

In a repository, a file can exist in one of four states:

1. Untracked
2. Unmodified
3. Modified
4. Staged



To see how this works, let's change that readme file to say something else, and then run git status again

```
$ nano fruit_list.txt
```

Let's change apple to kiwi:

```
banana
kiwi
peach
```

```
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:   fruit_list.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

If you want to see what changed, you can use `git diff fruit_list.txt`

```
diff --git a/fruit_list.txt b/fruit_list.txt
index b3231cb..2c3ec01 100644
--- a/fruit_list.txt
+++ b/fruit_list.txt
@@ -1,3 +1,3 @@
 banana
-apple
+kiwi
 peach
```

We can stage and commit this in one step with:

```
$ git commit -am "fruit_list updated"
```

Ignoring files

What if you have just put a bunch of files in your repo, that you want to add all at once?

```
$ touch LICENSE CITATION grapher.R sorter.py passwords.log
```

You could write them all out in your add command, or you can use `git add -A`

However, this will add *ALL THE THINGS* in your repo, which is probably something you don't want

> You don't want to clutter your tracked files with a bunch of temp files or OS garbage
> You *ESPECIALLY* don't want to accidentally share any keys or credentials files!

Right now, if you type `git status`, you'll see passwords.log show up as needing to be added. But you don't want that being shared on the repo, or on Github!

Luckily, git has a workaround for this called `.gitignore`

```
$ touch .gitignore
$ nano .gitignore
```

Let's make sure Git doesn't track any log files:

```
# Files to ignore
*.log
```

Now if we type `git status`, our passwords.log file is no longer displayed. It won't be added if we add all, and subsequently will not be committed.

We can check this by typing:

```
$ git status --ignored
```

Which shows:

```
On branch master
Ignored files:
  (use "git add -f <file>..." to include in what will be committed)

        passwords.log

nothing to commit, working directory clean
```

Thus we can safely proceed to add and commit all files:

```
$ git add -A && git commit -m "adding new files"
```

Acknowledgments

This learning module borrows and adapts materials from the following organizations and individuals. Thank you!

Software Carpentry (https://github.com/swcarpentry/git-novice)
Dav Clark (https://github.com/davclark/git-fundamentals)