1. INTRODUCTION

Git is a decentralized and distributed version control system.

While git is decentralized, most people still choose to use it with a central server to serve as the main repository for a project or team.
Because of the distributed nature, Git can scale massively.

For example, the creator of Git, Linus Torvalds, wanted to create a version control system

that could handle the requirements of the Linux Kernel project, which he also created.

Today, the Linux kernel project contains over 15 million lines of code,

with over 12,000 worldwide developers contributing to the project in dozens of active branches.

Another key benefit of being distributed is that most operations in Git are local.

There are only a few commands that require a network connection. Otherwise, you can work completely disconnected;

even preforming comparisons and commits completely off network. Since most operations are local,

git is very fast with the same operations. Also, Git is free and open-source,

which also contributes to its popularity. Git has a very active community,

and there are many resources available online.

Also, it is very easy to find developers that already have experience with Git.

Because of all these factors, Git has become the most popular version control system.

Git's De-Facto standard status means that Git enjoys wide adoption and integration into other tools used

by the development community, including text editors, bug tracking systems, and build servers.

Before we dive in, there are just a few concepts you need to know.
First, in Git, collections of version control files are kept together in a repository.

2. Key concepts:

- Repository contains files, repository, history and config managed by Git
- Three states of Git
  - Working directory
  - Staging area – Pre-commit holding area
  - Commit – Git repository (history)
- Remote repository (Github)
- Master branch

The repository also contains the history of changes, and any special configuration.

Generally speaking, a repository would contain all the files related to a specific project or application.

Next, you need to know about the three local states related to files being managed by Git:

the working directory, staging area, and the Git repository or the commit history.

The working directory is the directory or folder on your computer that holds all the project or application files.

Files within the working directory may or may not be managed by Git; however, Git is aware of them.

Normally, within the working directory is a hiden folder called the ".git" folder,

that contains the actual Git repository. The Git repository manages the Git commit history

that is, all the changes that are finalized and permanently part of the Git repository.

In-between is the Git staging area, often referred to as the Git index,

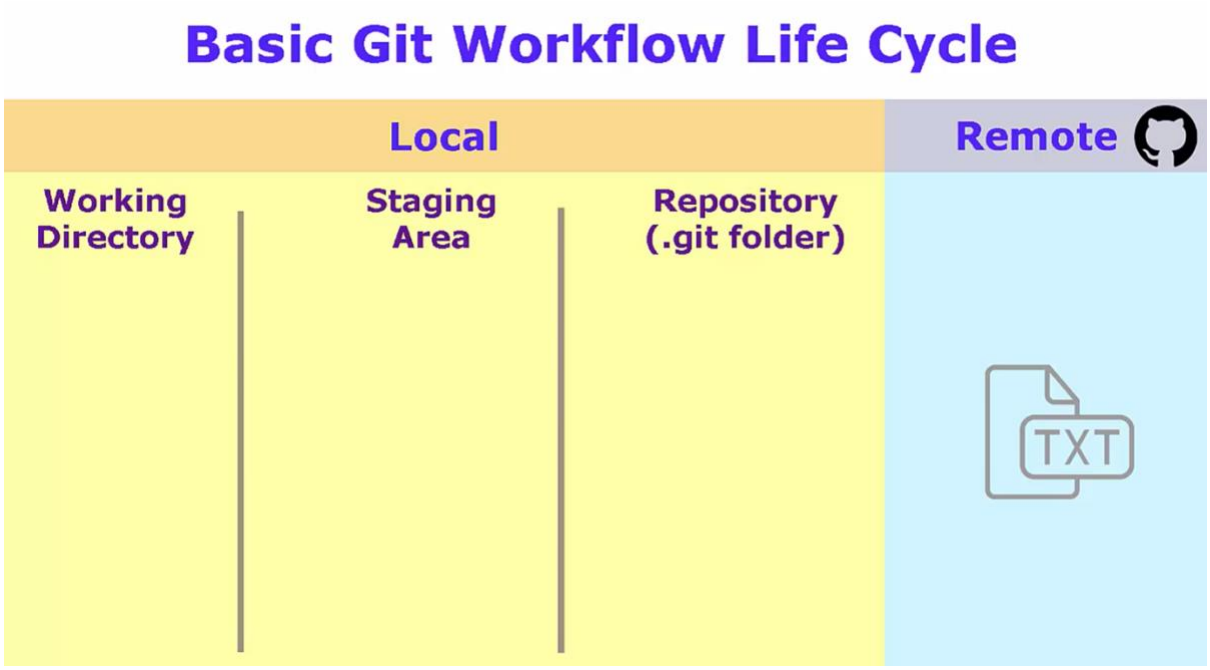
that is a holding area for queuing up changes for the next commit.

Since files in the staging area haven't been committed yet, you can move the files in and out of the staging area

without impacting the Git repository and its history of changes.

The three states of Git are specific to the local Git repository, but I like to tack on a fourth state: the remote state.

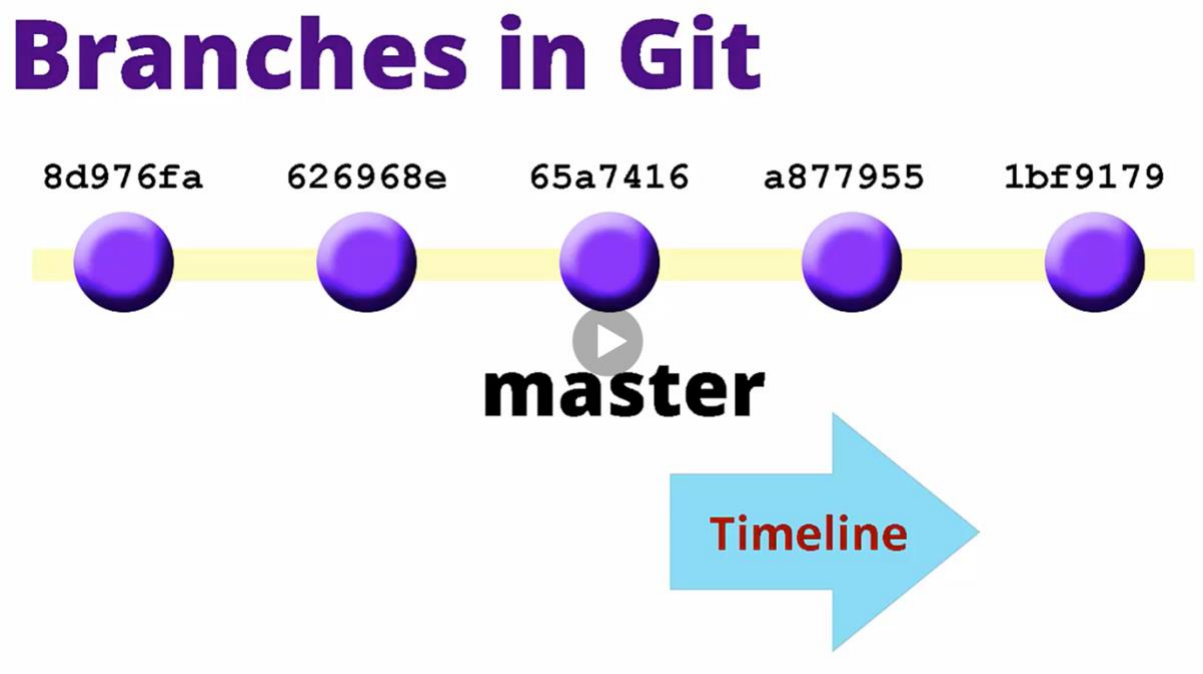
Although the remote repository is just another repository with its own three states internally,



conceptually, I think of the remote repository as a fourth state, since it is the last step in the basic Git workflow,

and since few people use Git without a corresponding remote repository.

The last concept is the master branch. Conceptually, branches work like they do in other source control systems:



they are timelines that contain your changes. In Git, branches contain commits.

When we start off, Git provides us with a default branch named master.

Great! That's all I wanted to cover at this point; I'll cover more concepts later and while I'm in the demos in the next videos.

3. Why COMMAND LINE

- History
- New Features
- Online help!
- Power
- Consistent
  - Terminal on Mac/Linux
  - Git Bash on Windows

The majority of my course will be done with the command line.

I certainly understand if you feel a bit intimidated by using the command line, especially if you have never used it before.

However; I will walk you through, step by step, each command along the way.

There is nothing wrong with using a graphical client; but I firmly believe starting off with the command line is the best way to learn Git.

So, you might wonder why, or perhaps even think the command line is a step backwards.

Here are my reasons; first, there is a strong history with the command line in Git.

It was originally designed as a command line tool, and graphical clients came later.

As a result, new features make it onto the command line well before they are integrated into a graphical client.

For me, one of the most compelling reasons is the online help:

Nearly all online assistance, websites, blogs, or other tutorials use the command line

as the standard way of communicating how Git does something.

There are dozens of graphical clients, but everyone has the same commands in Git.

With only a few fringe exceptions. Also, the command line has more power.

Most graphical clients only implement the core or common commands,

but leave off some very powerful options. Only the command line provides all the raw power of Git.

Much like my online help argument,
going command line allows me one set of commands in Git that will work on Windows, the Mac OS, and even Linux.

I will point out any commands, Git or otherwise, that are unique to a platform.

4. INSTALLATION ON WINDOWS

In this video, we're going to install Git for Windows.

There are actually several ways to install Git on windows. We are going to use a project called "Git for WIndows",

that provides us the most up-to-date version available for the Windows platform.

This project is an improvement over the older recommendation of using the git-scm.com version of Git for Windows.

As of the recording of this video, Git for Windows is still in preview; however, it is very stable,

and for our purposes will be just fine. Either way, I recommend downloading and installing the latest version of Git for Windows.

Let's start off by invoking our browser, in this case Google Chrome, then go to "git-for-windows.github.io".

Once that page loads, click on the download button on the home page. This brings us to a page listing all the available downloads

for the GIt for Windows installer. You can see here that we have both 32 bit and 64 bit versions of the installer executable.

Since I'm running a 64 bit version of Windows, I'm going to choose the 64 bit version of Git for Windows.

Choose the version of Git for Windows that best matches your operating system.

Now I'm going to click on the link for the Git installer for my platform. That will begin the download process for the Git installer.

Once the installer has finished downloading, go down and click on "Open".

That will run the installer directly from the current location. If you are prompted about a security warning,
just click on run. Windows may also prompt you to make sure that you want to install this program. Click on "Yes".

Once the Git setup wizard starts, on the first page, click on "Next". The second page has the license agreement.

Again, click on "Next" to agree to the license agreement. The next page is select your destination.

This is the location that Git will be installed on your system. For me this location is perfectly fine, so now I will click on "Next".

On the "Select Components" page, I would like the Git Bash icon on the quick launch, or taskbar, as well as on the desktop.

To get both of those icons I will check "Additional icons", which also checks "In the Quick Launch" as well as "On the Desktop". Then click on "Next".

This is the folder on the start menu that Git will be installed. Since this is fine I will continue by clicking on "Next".

This page dictates what parts of Git and GIt Bash are installed. Although we will only be using Git Bash as part of this course,

I recommend selecting the middle option. This will install Git Bash, as well as integration with the Windows command prompt.

In doing so, no real major changes are being made to my system. So it is both flexible and fairly safe.

After you've made your choice, click on the "Next" button to continue. This screen is regarding how to treat line endings.

On Windows, text files normally end with a different style line ending than corresponding text files on Linux or the Mac operating system,

which option you choose will greatly depend on whether or not you are going to be doing any cross-platform development.

I prefer using the middle option: which is to checkout as-is, that is don't change anything, but then commit Unix-style.

That means, when I use Git to commit, the file endings for my text files will automatically be converted to a Unix-style line ending.

Since I bounce around between Windows, Linux, and the Mac operating system, the middle option makes the most sense for me.

If you do all your development on Windows, and that goes for any team members sharing the same Git repositories,

then the last option, "commit as-is" will be the option you'll want to select. Again I'm sticking with the middle option,

and if I want to later, this can be changed. Once we've made our selection, click on on next.

Regarding the terminal emulator to use, I'm going to stick with the top option, which is a minimal terminal emulator that runs the bash shell environment.

Once you've made your selection, click on "Next". This page, "Enable file system caching" is experimental,

and I recommend leaving it unchecked for now. Click on "Next", and now we are installing Git for Windows.

Once the installation process has completed, on the final page we can un-check to view the README.

Then click on the "Finish" Button. Now, let's close Google Chrome. While we're here, let's verify that Git for Windows has an icon on our desktop.

Looks like it's there. So now let's double click on Git Bash to fire up Git Bash for the first time.

Now that we've done that, let's verify that Git is installed. Type git, that's g-i-t, space, version. Once you've done that press the enter key.

Git should respond with the version of Git that's installed.

To close Git Bash, we can click on the close button that's part of the window, or we can type the command "exit".

5. INSTALLING ON MAC

Welcome, in this video, we're going to install Git on the mac.

Open up your terminal program, and simply type "git version". Then press enter.
If you get this response, that means that Git is not installed; However, with newer version of the Mac operating system

we will be automatically prompted to install Git using the developer command line tools.

When this prompt comes up, you have the option of getting Xcode, which will include Git; dismiss this dialogue entirely,

which means not to install Git; or to install, which will install Git using the command line tools.

Click on "Install". On the license agreement, click on "Agree". This will being the download and installation process.

Once the installation process has finished, click on done. Great, to make sure we have a clean terminal session,

let's quit out of terminal, command+q, and fire it back up. Now type "git version", then press enter,

if our installation was successful, Git will respond with the version number that has been installed.

And in this particular case, we have, in parenthesis, the qualifier that it's an Apple version of Git.

6. SETTING UP PROJECT FOLDER

Now that we've created our first repository on GitHub, we need to prepare our local system before we can proceed.

I'm going to be doing most of my work in the command line. On Windows, use Git Bash.

On the Mac, use the default terminal application, or a third party terminal program like iTerm 2,

which is what I'm using. Either way the Git commands will be the same.

So now, open your terminal program. By default, your terminal program should put you in your user's home directory.

Which we can confirm by typing "pwd". I like to manage all my projects together,

in a projects directory within my user's directory. To create that directory type:

"mkdir projects" then press enter. Then change into that directory.

"cd projects", press enter, now if we "pwd",

You can see that we are now within my our projects folder underneath the user's home directory.

7. GIT CONFIGURATION (USER NAME AND EMAIL)

On Windows, if you installed Git for Windows, and you are using Git Bash, then you know Git is installed.

On the Mac, it is less obvious, so we need to ask Git for the version in order to confirm that Git is available:

"git version". If Git responds with a version number, then Git is installed.

Otherwise you may need to install Git before proceeding.

Git requires two bits of information before we can do very much: that is the name and email address.

If we don't provide that information, Git will attempt to automatically figure it out.

However; I have found this process to be rather unreliable. So let's just take care of it right now:

"git config --global user.name ", and in double quotes just put your name.
And then next, we'll do a similar command for the email address:

"git config --global user.email ", and in double quotes put your email address.

Press enter, and now let's confirm: "git config --global --list".

Git should list back your name and email address you entered above.

8. COPY THE REPO FROM GITHUB TO LOCAL (GIT CLONE)

Now that our local system is all ready to go, let's return to our browser where we left off,

which should be in our new repository's main page. Now I want to get a copy of this repository onto my local system.

The process to do that is called cloning a repository.

So, locate the clone options at the bottom of the right-hand side of the page.

Make sure HTTPS is selected. And then click the copy button to copy the HTTPS URL for the repository

to your system's clipboard. If, for some reason, that button isn't available or isn't working,

just select the HTTPS URL and then copy it into your clipboard manually.

Now that's done, return back to the terminal and type "git clone ", and now just paste in your GitHub URL.

Double check that line, and then press enter. Git will now go out to GitHub,

and make a full copy of our repository on GitHub to our local system.

Doing so, Git will automatically create a directory named after our repository.

Which we can confirm by typing "ls", then press enter, which lists out the contents of our current directory.

Now, let's go into our local repository's directory: "cd github-demo/"

Now if we "ls", we can see we have our README file that was created when we initialized

our repository on GitHub. Now if I ask Git about the state of the repository with "git status",

Git status tells us that I'm on the master branch.

The master branch is the default branch, by convention, for a Git repository.

The "git status" command also tells us master is up-to-date with 'origin/master',

which refers to the master branch on the GitHub version of the repository.

The git clone command automatically set up a relationship back to the repository on GitHub

and named that reference origin. Finally, Git tells us the working directory is clean.

The working directory is where we do all the work and that Git monitors for changes.

We use the "git status" command to see if there are any changes between the working directory,

the staging area, our local repository, and our remote repository.

git clone <https: url>

9. THE FIRST COMMIT

Now that we have our repository on our local system, let's add a new file.

For that, I'm going to use the bash commands to create a simple text file.

So type: echo; space; and in double quotes the contents of our file

"Test Git Quick Start demo"; and then two greater than signs

that pipes contents of the echo command into a "start.txt" file.

After that, press enter. Now we can see the file in our working directory: "ls".

Also, we can use the cat command to display the contents of that file:

"cat start.txt" press enter. Now, let's see what Git thinks about this file:

"git status", press enter. Now when we do that Git says that we have an untracked file.

An untracked file is just a file in our working directory that hasn't been added to Git yet.

Our change, that is our new file, is in our working directory.

We haven't specifically told Git about it yet, so let's take care of that now with Git's add command:

"git add ", and then the name of the file: "start.txt".

Press enter; now if we do a "git status", Git tells us there is a new file in the staging area,

which Git describes as changes to be committed.

So, now that our file is in the staging area awaiting the commit, we haven't committed anything yet.
We can still back out the change from the staging area, and not impact Git's history in any way.

The staging area is designed to allow you to build up several related changes,

so they can be committed as a single atomic unit.

Now, let's move forward by committing the new file into the repository:

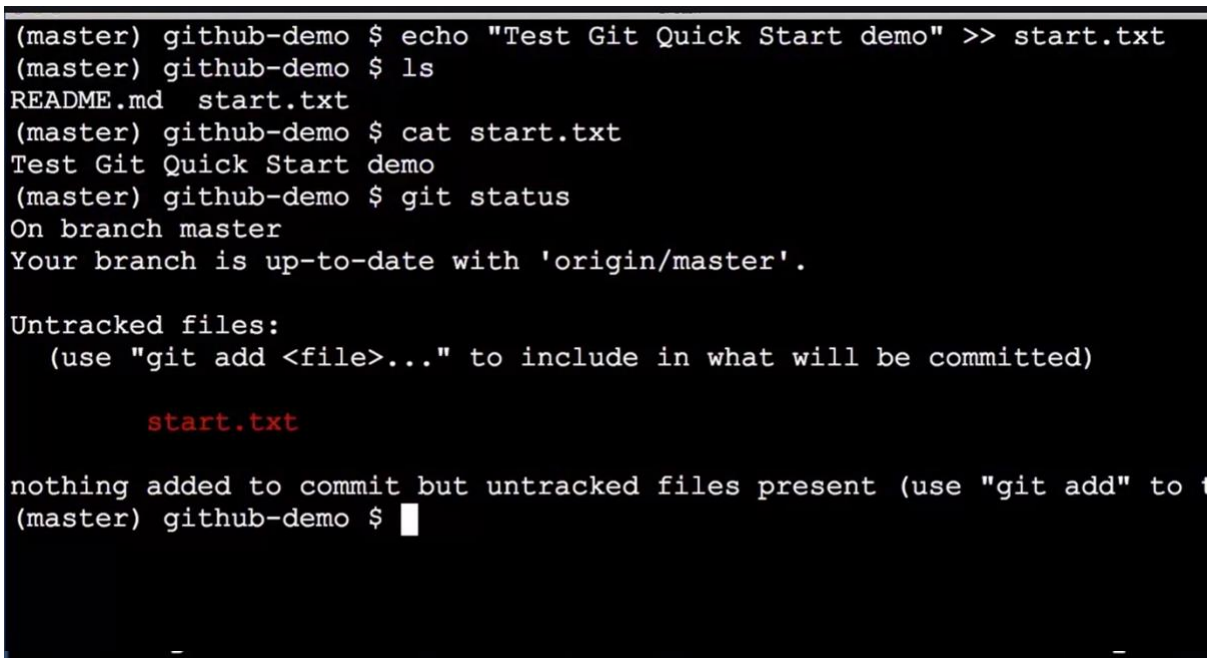
"git commit -m ", and in double quotes a commit message: "Adding start text file", then press enter.

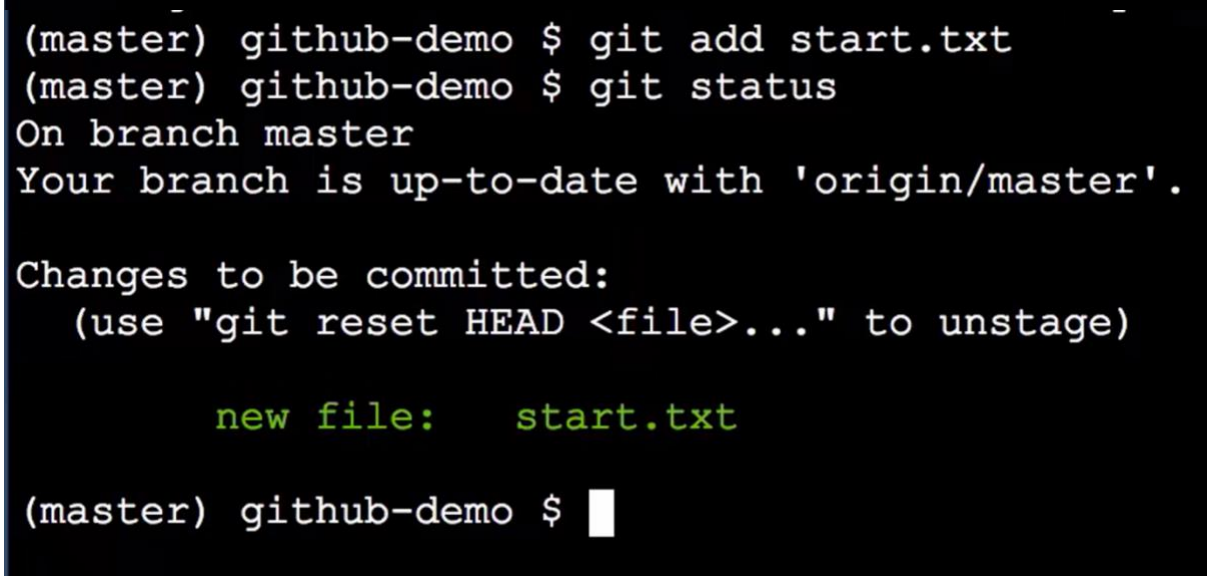
Great, now if I do a "git status", Git tells us that we're back in a clean working directory state,

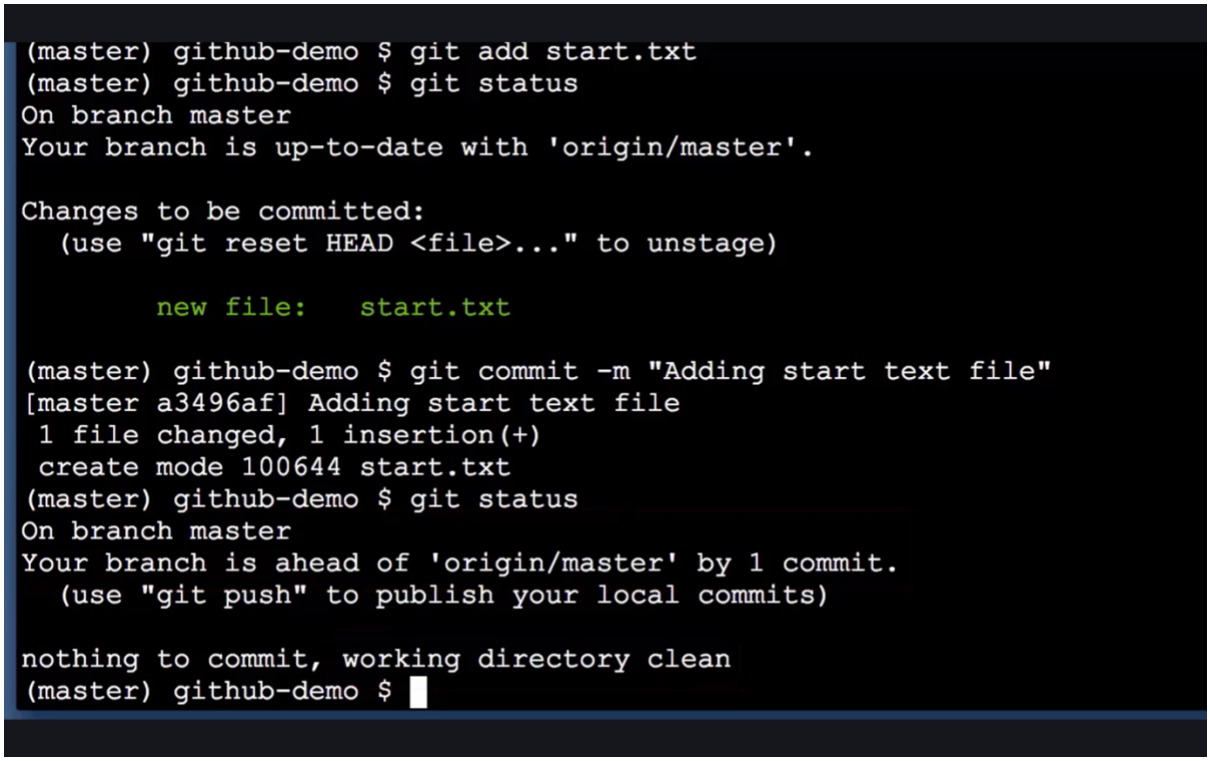
and that our master branch is ahead of origin master by one commit.

So, now the new file has been moved from the staging area, into the local repository.

Since there are no other pending changes, once the commit happens Git marks the working directory as clean.


(master) github-demo $ echo "Test Git Quick Start demo" >> start.txt
(master) github-demo $ ls
README.md start.txt
(master) github-demo $ cat start.txt
Test Git Quick Start demo
(master) github-demo $ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
(use "git add <file>..." to include in what will be committed)
start.txt
nothing added to commit but untracked files present (use "git add" to
(master) github-demo $


(master) github-demo $ git add start.txt
(master) github-demo $ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)
new file: start.txt
(master) github-demo $



10. PUBLISHING CHANGES BACK TO GITHUB (PUSH)

After the commit, I mentioned the fact that our new file is now on the local repository.

The commit is still a local command; which means that our file is not yet on GitHub.

So to prove this, let's return to our browser. We should still be on our repository main page;
so refresh or reload your browser to refresh the contents of that page.

What you should notice is that we still have just the README file on GitHub.

So there is one last step we need to do, and that is a push. So return to the terminal,

and type: "git push origin master". This version of the "git push" command

specifies the remote name, which is origin,

which was set up for us automatically when we cloned the repository from GitHub;

and the name of the branch to push, which is master, since it is the only branch we have.

Now press enter. Since you are making changes on your repository on GitHub,

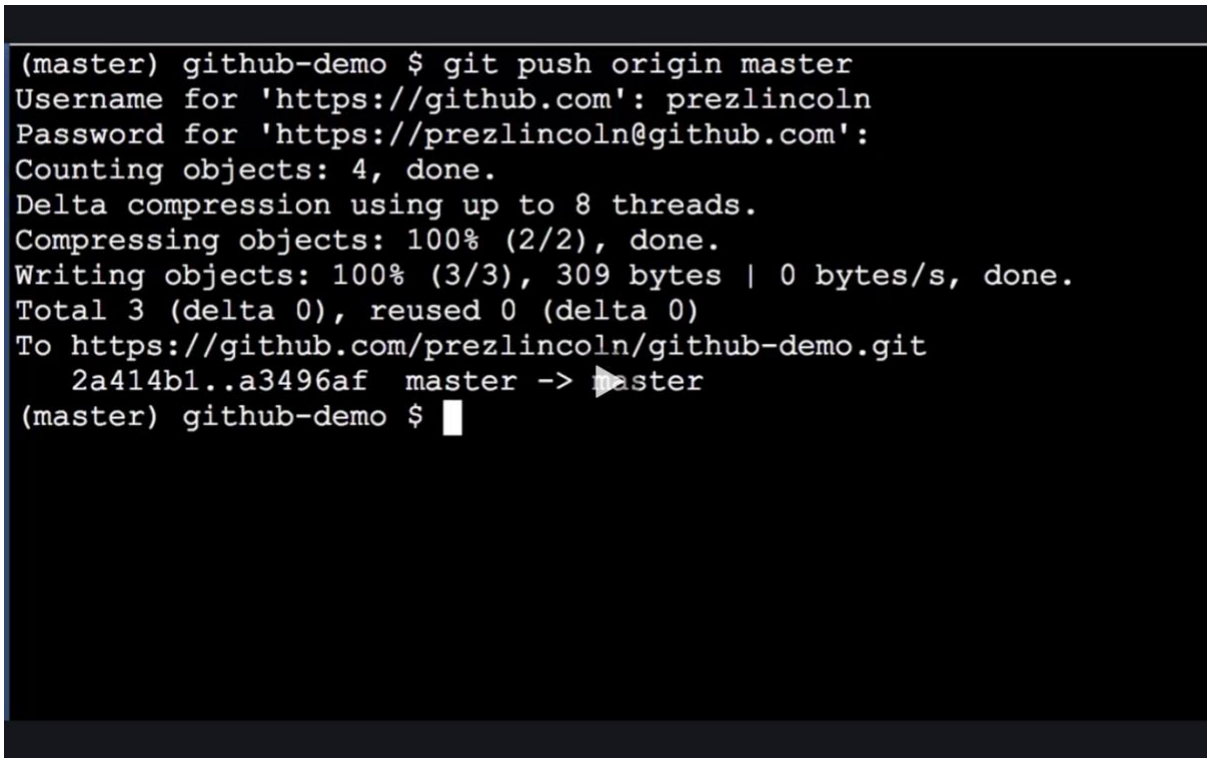
the "git push" command will prompt you for your GitHub username and password.

Once the push command returns, then we can check the results.

If we did everything correctly, our new file should be on the GitHub copy of our repository.

Let's verify that, by returning to our browser; now refresh the page again.

Now, we should see our "start.txt" file listed. If I click on the filename, I can see the contents of that file as well.



11. CREATING A NEW REPO LOCALLY

In this video, I'm going to cover how to start a new project without any existing source code.

As part of this demo, I'm going to be using some filler text from the hipster ipsum website,

which is available at "hipsum.co".

So to start off, I'm going to return to my terminal, and I'm starting off in my user's home directory.

"pwd" shows that I'm in the root folder of my user's home directory.
Within there, I have a projects folder. So "cd projects", and here I want to create a fresh new Git repository.

To do that I'm going to use Git's "init" command. So "git init ", and then the name of your project.

For this example, I'll call it "fresh-project". Now press enter.

In doing so, Git created a folder called "fresh-project". So if I do an "ls",

you can see that I have a "fresh-project" folder. Alright so let's go into it:

so "cd fresh-project/", now press enter.

I'll go ahead and point out a few important pieces of information here; if I do an "ls",

you'll see there are no new files or folders listed. However, if I do an "ls -al";

that tells the list command to list all files and folders, including dot files and folders,

and "-l" tells it to list in the listing format. Now you can see that we have a ".git" folder.

That is actually where the Git repository lives. The folder that I'm in now is actually the working folder.

So if I go into the ".git' folder, you can do an "ls", you can see all the various folders and files

that make up a Git repository internally. So if I go back,

now I'm back in the working directory, and I'm going to do one more thing; I'm going to use Git's "status" command,

which will let me know the status of the Git repository. So type "git status",

press enter, and the "git status" command tells us that we're on the master branch.

Now, as described before, the master branch is the default branch that Git gives us in the beginning.

We can always create more branches later, but for now we'll stay on master.

Git tells me that I'm on the initial commit, and that I have nothing to commit yet.

Now that's because I haven't created any files yet. Well let's go ahead and do that.

So if I head back over to my browser and you can see that I'm on the Hipster Ipsum website.

Four paragraphs is fine; we're probably just going to use the first one.

And it really doesn't matter which option you choose here: we just need filler text.

So "Beer me!' is the way that we get the text. So I'm going to just take that first paragraph,

control click, or right click and then copy. That puts that text into the clipboard,

and now I'm going to return to my command prompt. Now let's create a new file:

So "mate ", and then the name of a file: I'm just going to create it "hipster.txt",

now press enter. That fires off TextMate.

And now we should just be able to press command+v to paste in all that ipsum lorem.

Great. Now let's save by doing command+s, and close: command+w.

Now that we've returned back to the command prompt, now if I do a "git status";

again, Git reminds us that we're on the master branch, and that we're building up our initial commit.

However, we have some additional information; Git tells me that I have some untracked files,

and it sees the "hipster.txt" file. Well we're currently in the working directory, so Git is not tracking this yet.

To have Git track this file, we simply need to add that file to the Git index, or to GIt's staging area.

To do that, we simply type "git add ", and then the file that we wish to add. So now I'm going to press enter.

Now if we do our "git status", Git tells us that we have changes to be committed.

That means this new file, "hipster.txt", is now in the Git staging area.

We have a file that's waiting to be committed, but it's not committed yet.

We could back out this file, and it's not going to impact Git's history in any way.

However, at this point, I do want to move forward with this file.

So now if we type "git commit" without any extra parameters,

Git will invoke our default editor, which it currently is TextMate, for our commit message.

So now i can type my commit message: so "Adding new file with hipster ipsum",

and on a second line, "this was done with Text Mate 2".

So one advantage of using the text editor for my commit messages, is that I have the option of doing multiple line commit messages.

Now to use this as my commit message, I will save and then close.

Command+s, and then command+w. So you can see that the commit happened

and that it used the commit message that I entered into the text editor.

Now one thing to note is that when the command prompt returns with the commit message,

that all the lines are run together; so it's not going to preserve the new line character,

but that only happens as part of the Git commit command.

Those newline characters are being preserved and I'll show you that later.

Additionally, you'll notice that in parenthesis, we have "(root-commit)".

That's just letting us know this was the very first commit of this repository.

So now, if I do a Git status, Git reminds me that I'm on the master branch.

And now I have nothing to commit, and that my working directory is clean.

So at this point my new file, "hipster.txt", is in the Git repository.

So now that change is part of the Git's commit history. Alright, well we're done for now.

So I'm going to back up a level; that brings us back to the projects folder.

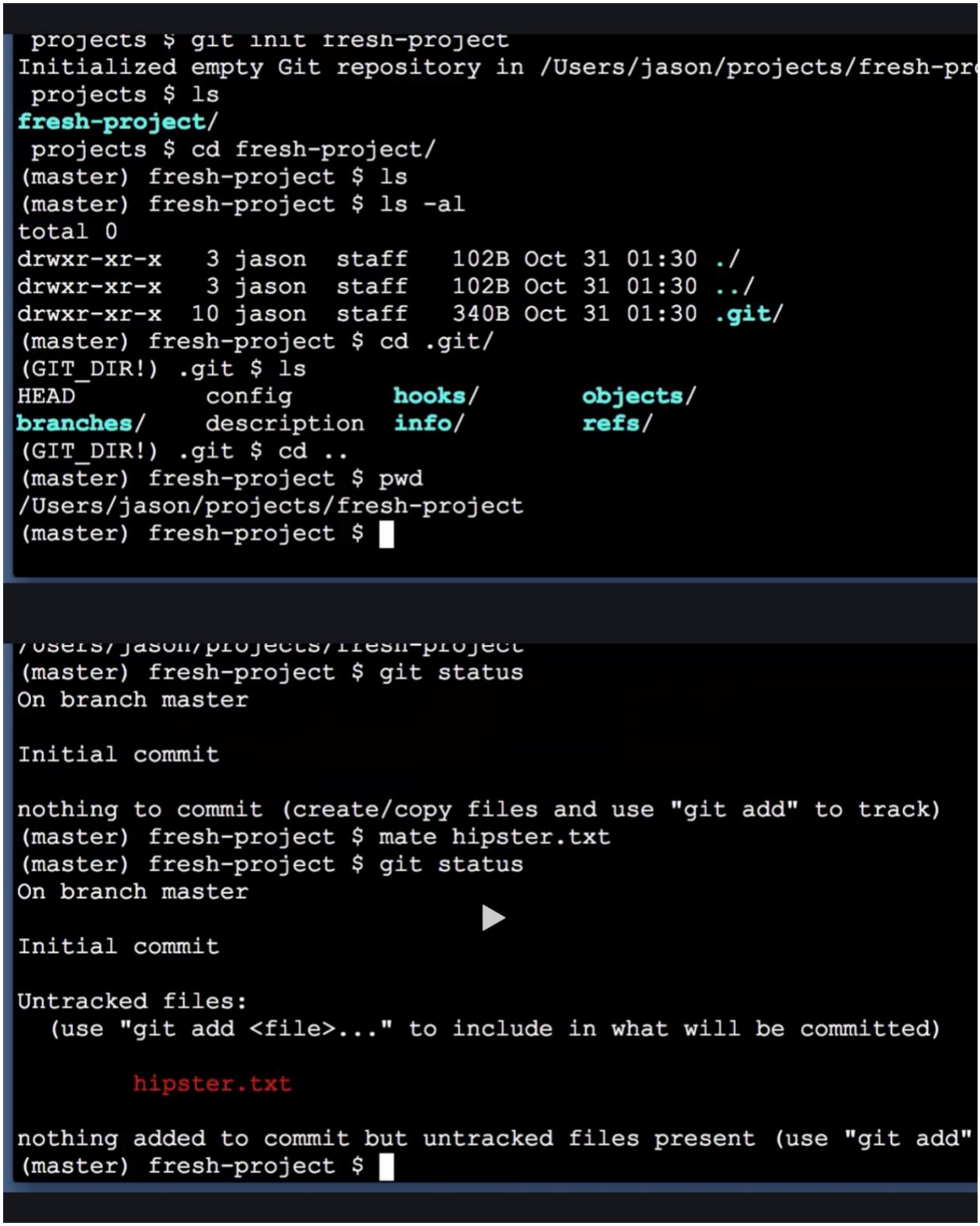
Do an "ls", you can see my fresh project is there; and now if I do a "rm -rf",

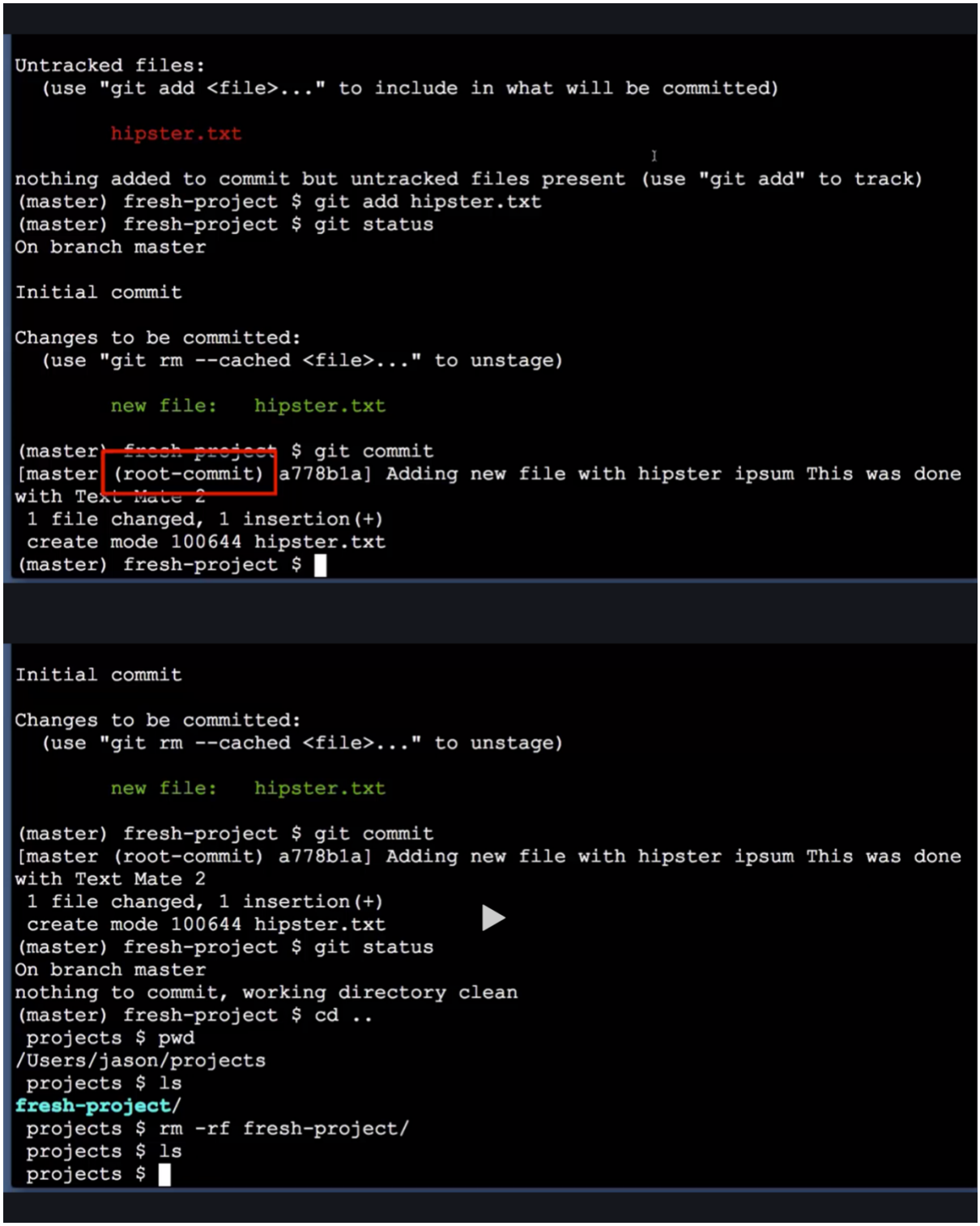
specify the "fresh-project/", it will delete that folder for me. So no more Git repository.

```
(master) github-demo $ git push origin master
Username for 'https://github.com': prezlincoln
Password for 'https://prezlincoln@github.com':
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 309 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/prezlincoln/github-demo.git
   2a414b1..a3496af  master -> master
(master) github-demo $
```

17. Create a New Repository Locally

```
 ~ $ pwd
/Users/jason
 ~ $ cd projects/
 projects $ pwd
/Users/jason/projects
 projects $ git init fresh-project
Initialized empty Git repository in /Users/jason/projects/
 projects $ ls
fresh-project/
 projects $ cd fresh-project/
(master) fresh-project $ ls
(master) fresh-project $ ls -al
```


projects $ git init fresh-project
Initialized empty Git repository in /Users/jason/projects/fresh-pr
projects $ ls
fresh-project/
projects $ cd fresh-project/
(master) fresh-project $ ls
(master) fresh-project $ ls -al
total 0
drwxr-xr-x 3 jason staff 102B Oct 31 01:30 ./
drwxr-xr-x 3 jason staff 102B Oct 31 01:30 ../
drwxr-xr-x 10 jason staff 340B Oct 31 01:30 .git/
(master) fresh-project $ cd .git/
(GIT_DIR!) .git $ ls
HEAD config hooks/ objects/
branches/ description info/ refs/
(GIT_DIR!) .git $ cd ..
(master) fresh-project $ pwd
/Users/jason/projects/fresh-project
(master) fresh-project $
(master) fresh-project $ git status
On branch master
Initial commit
nothing to commit (create/copy files and use "git add" to track)
(master) fresh-project $ mate hipster.txt
(master) fresh-project $ git status
On branch master
Initial commit
Untracked files:
(use "git add <file>..." to include in what will be committed)
hipster.txt
nothing added to commit but untracked files present (use "git add"
(master) fresh-project $



12. ADDING GIT TO EXISTING REPO

Welcome, in this video, I'm going to demonstrate how to add Git to an existing project.

To do so, I'm going to set up a new project using Initializr, which will give me a bunch of source code,

and then, I'm going to add Git source control to that project.

So point your favorite browser to "initializr.com".

Once that page loads, for this demo you can pretty much use any configuration.

I'm going to use Bootstrap, and then I'm going to leave this as default.

Then I'm going to add all these options just to give me some more files.

Once I'm done, I'm going to click on "Download it!".

Alright, great; now that it's downloaded, I'm going to make sure it's in my downloads stack.

And there it is, initializr dot zip. So now I'm going to return to my terminal,

and I'm currently in my user's home directory; so "pwd' shows that I'm in my root folder in my user's home directory.

I'm going to go into my projects folder, and here I'm going to expand that zip file that i just downloaded.

So "unzip ~", which is the way we reference our user's home directory,

"/downloads/", and then the name of the zip file. Once you have that, press enter.

So the "unzip" command should expand everything out. So now if we use the "ls" command,

you should see that we have our initializr folder. So let's rename this folder so it looks more like a real project.

So I'm going to use the bash command, "mv", to move the folder, which is another way of renaming,

specifying the initializr folder, and giving it a new name, "web-project".

Now press enter; alright, great, so if I do an "ls" you can see that I have a "web-project" there.

If I go into it, "cd web-project/", press enter.

Now I'm in the web-project folder, so if I do a "pwd" you can see where I'm at.

And, if I do an "ls", you can see all the files that we got by unzipping that zip file.

Great, so I'm going to pretend that this is my project and I need to add Git source control

to this project. So to do that I can use Git's "init" command.

So if i type "git init" and then press enter,

Git will initialize a new repository, using the current folder as the current working directory.

Now if I do an "ls -al", I can see that I have my ".git" folder,

which is where the Git repository actually lives, along with the other files that are part of this project.

Now if I do a "git status", Git will remind me that I'm on the master branch,

which is the default branch that you get when you create a new Git repository.

And since we haven't done any commits yet, we are building up to our initial commit.

And then Git notices a bunch of files that it's not tracking yet.

So, in order to add all these files at once to Git's staging area and to the git index,

we're going to type "git add " and then a period, then we can press enter.

Now if we do a "git status", Git lists all the files that are part of this project.

Now there are a lot more files than we have before,

and that's because the period will recursively add all the files that are part of this project.

Well great, let's continue on by committing. So type "git commit " and now I want to provide a commit message inline;

so that would be "-m ", and in double quotes, put your commit message.

I'm using "My first commit, inline", now press enter.

So, in this case, the commit command used the commit message we provided

with the "-m" option, instead of invoking our text editor.

I like using this approach if I just have a one-liner if I want to use as my commit message.

So let's do our "git status". And, as expected, after a commit

we have nothing to commit, because we are on a clean working directory.

So I'm going to show one last thing. So if i clear, you'll notice that I have in parenthesis "(master)".

And if I do a "git status", it says that I'm on the master branch, and that I have nothing to commit,

and that my working directory is clean. That's because this working directory is part of a Git repository or being managed by a Git repository,

which is being maintained by that ".git" folder. Again, if I do an "ls" with an "-al" option,

you can see that I have that ".git" folder. If that ".git" folder was no longer there,

this project would no longer be managed by Git. So, let's do that.

So now if I type "rm -rf ", and then the folder ".git", then press enter.

We just deleted the ".git" folder, and in doing so we removed all traces of Git managing this project.

If I do an "ls", you can see that we still have all the files that were part of the original zip file.

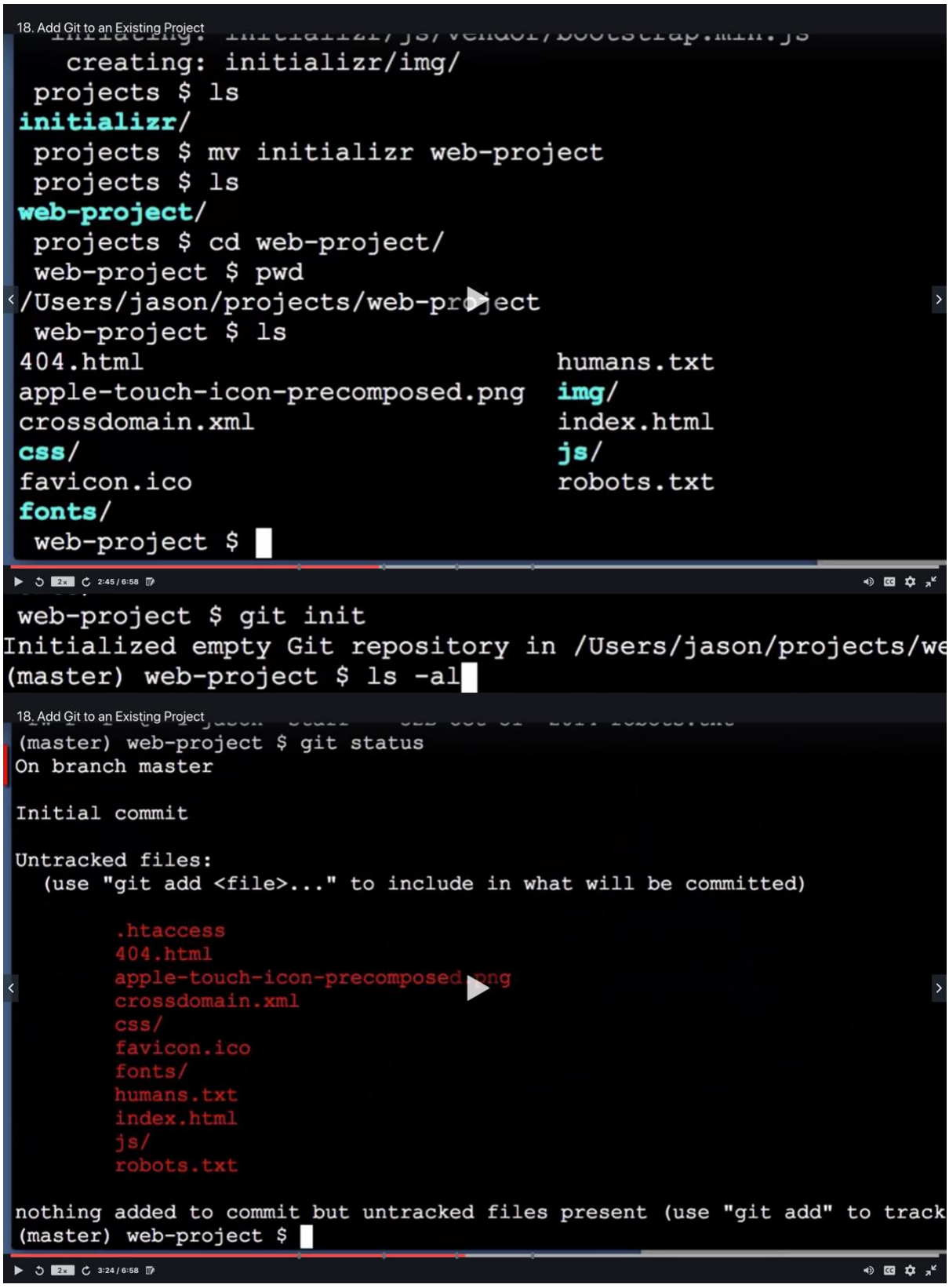
Well since I'm done with this example, I'm going to clean up after myself. So I'm going to back up a level,

so now you can see where I'm at: I'm just within the projects folder. Do an "ls",

you can see the "web-project" directory. So now let's remove it.

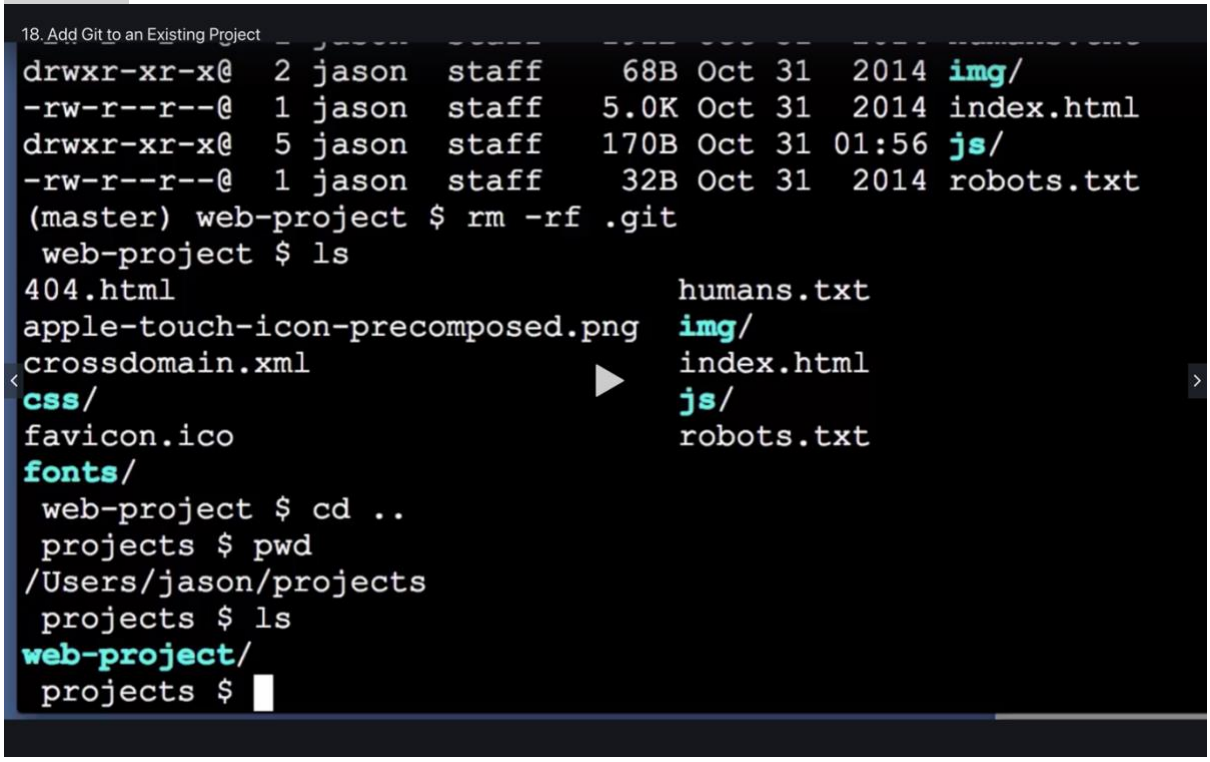
So type "rm -rf ", and then the name of the folder:

"web-project". So now press enter, and we just removed our project. "ls" and nothing is there.




(master) web-project $ git add .
(master) web-project $ git status
(master) web-project $ git commit -m "My first commit, inline"

Git status


18. Add Git to an Existing Project
drwxr-xr-x@ 2 jason staff 68B Oct 31 2014 img/
-rw-r--r--@ 1 jason staff 5.0K Oct 31 2014 index.html
drwxr-xr-x@ 5 jason staff 170B Oct 31 01:56 js/
-rw-r--r--@ 1 jason staff 32B Oct 31 2014 robots.txt
(master) web-project $ rm -rf .git
web-project $ ls
404.html
apple-touch-icon-precomposed.png
crossdomain.xml
css/
favicon.ico
fonts/
humans.txt
img/
index.html
js/
robots.txt
web-project $ cd ..
projects $ pwd
/Users/jason/projects
projects $ ls
web-project/
projects $