INTRODUCTION

1. Welcome
2. Audience and Approach
3. About the Instructor
4. Why Source Control
5. How Git Works
6. Command Line Start

Installation

1. Overview
2. System Setup
3. Chrome
4. Command Line
5. Git for Windows
6. Git Help
7. Git Configuration
8. Installation Notes

Git Bash

1. Overview
2. Getting Around
3. Command Help
4. Files and Folders
5. Vi Survival Guid
6. Projects

Quick Start

1. Overview
2. GitHub Account
3. GitHub Repository
4. Clone
5. First Local Commit
6. Edit Files
7. Push Back to GitHub

Command Shell

1. Overview
2. Command Prompt
3. Notepad
4. Start Locally
5. First Commit
6. Going Remote
7. PowerShell

Text Editor

1. Overview
2. Atom Install
3. Atom Fonts
4. Atom with Git
5. Using Atom
6. Atom and Windows

Basics

1. Overview
2. Setup SSH
3. Working with Remotes
4. Start Local with an Existing Project
5. Recursive Add
6. Delete
7. Rename
8. Git Log
9. Aliases
10. Ignore Unwanted File
11. Backout Changes
12. Going Remote to GitHub

Compare

1. Overview
2. Install Diff and Merge Tool
3. The Setup
4. Comparing Differences
5. Compare Previous Changes
6. Git Show
7. Compare Previous Commits
8. Compare Remote Changes
9. Just a bit of Cleanup

Branching

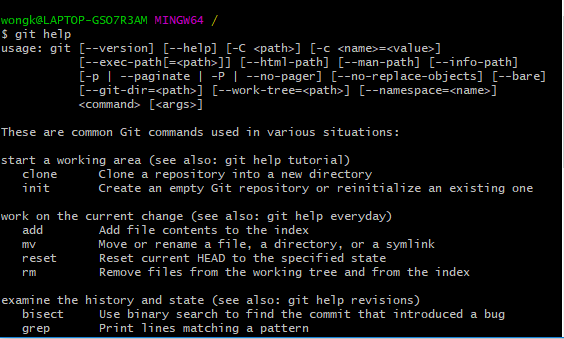
1. Overview
2. Easy Branching
3. No Fast Forward Merges
4. Fasf
5. As
6. dfadsf
7. dfadsf
8. Fasf
9. As
10. dfadsf
11. Fasf
12. As
13. dfadsf
14. Fasf
15. As
16. dfadsf

**LECTURE 4 – Why Source Control**

* Git is the leading tool for **Decentralized/Distributed** version control.
* **GitHub** is an online hosting provider for Git Repositories.

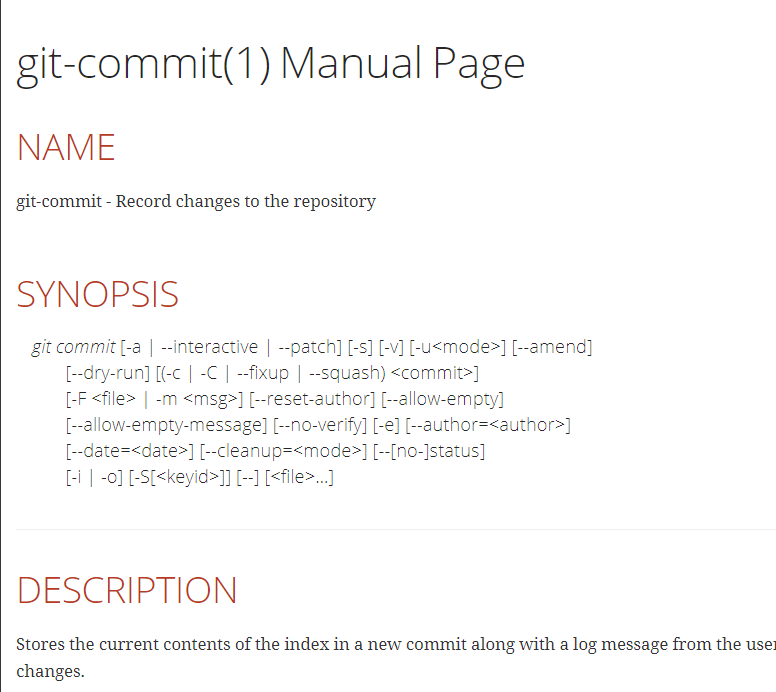
**LECTURE 12 – Git Help**

* To get access Git Help, in GitBash type **git help**.

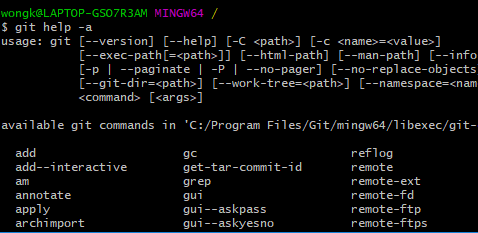


* To get more help on a specific sub-command, type **git help <sub-command>**. A web page will appear in your default browser with information related to that sub-command.





* To get a list of all the sub-commands, type **git help -a**.



**LECTURE 13 – Git Configuration**

**This is located upstairs**

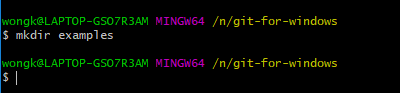
**LECTURE 16 – Getting Around**

* Here we are going to learn about a few windows commands to navigate through the directory.
* **pwd** – Stands for **print working directory**. This tells you which directory you are currently in.
* **cd** – Navigates to a specified directory.
* **cd ~** - Navigates to the User’s home directory.
* **cd /C** – Navigate to the C drive.
* **ls** – Lists all the files in the current directory.
* **ls -l** – Lists the files in a more detailed format.
* **clear** – Clears the terminal
* **exit** - Exit the terminal.

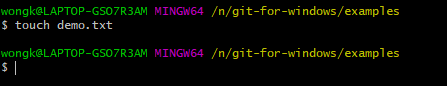
**LECTURE 17 – Command Help**

**LECTURE 18 – Files and Folders**

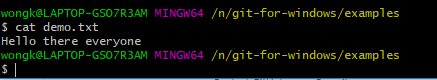
* Create a folder for your git-for-windows stuff.
* In Git Bash navigate to that folder.
* Use the following command in the Git Bash terminal to create a new directory called **examples**:
* **mkdir <name of directory>**.



* In Git Bash navigate to the **examples** directory.
* We are now going to create a new file using Git Bash.
* In Git Bash type **touch <name of file>** to create a new file.



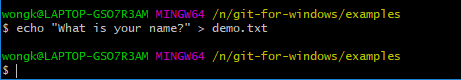
* To display the contents of a file on Git Bash, use the following command.
* **cat <name of file>**



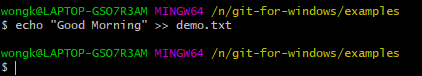
* To display text on the screen use **echo**, just as follows.
* **echo <Text to display>**



* To print text from the Git Bash terminal to a text file use the following command. This will replace all the existing text in the document with the new text.
* **echo <Content to print> > <File to print to>**

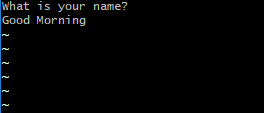


* To **append** text to a text file use the following.
* **echo <Content to print> >> <File to print to>**

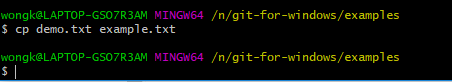


* To use a Text Editor in Git Bash and print the text to a text file, use the following.
* **less <Name of File>**
* Type the **q** character to get out of the Git Bash Text Editor.

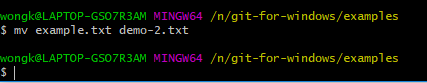




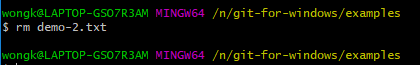
* To copy a file use the following.
* **cp <Name of Source File> <Name of Destination File>**



* To move/rename a file use the following.
* **mv <File to rename> <Name of new file>**



* To remove a file use:
* **rm <Name of file to delete>**



* There are a couple more commands to know.
* **rmdir** – Remove an empty Directory
* **rm -rf** – Remove a Directory and all the contents within the directory.

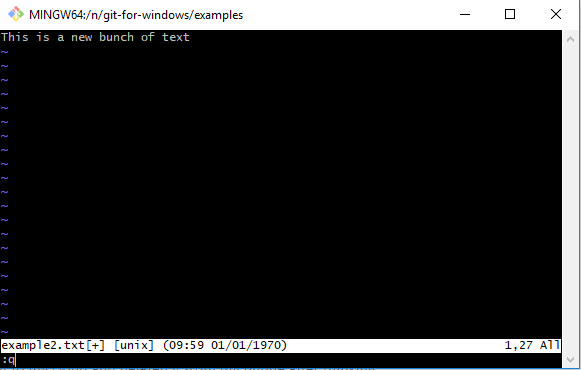
**LECTURE 19 – Vi Survival Guide**

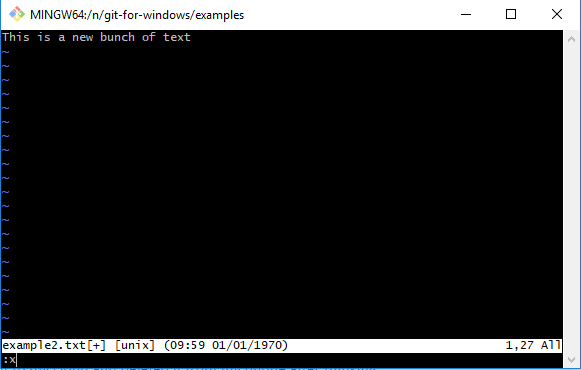
* Git has it’s own in built editor called **vi.**
* The **vi** command will open an existing file for editing or create a new file for editing.
* To use this type the following command.
* **vi <Name of file to edit or create>**





* Press **I** to enter **Insert Mode** which enables you to input text into the file. Press **ESC** to exit Insert Mode and enter the **Command Mode** which is where vi enables you to enter commands.
* The following is a couple commands in the vi command mode.
* **:q** – Exit without saving.
* **:x** – Exit with saving.



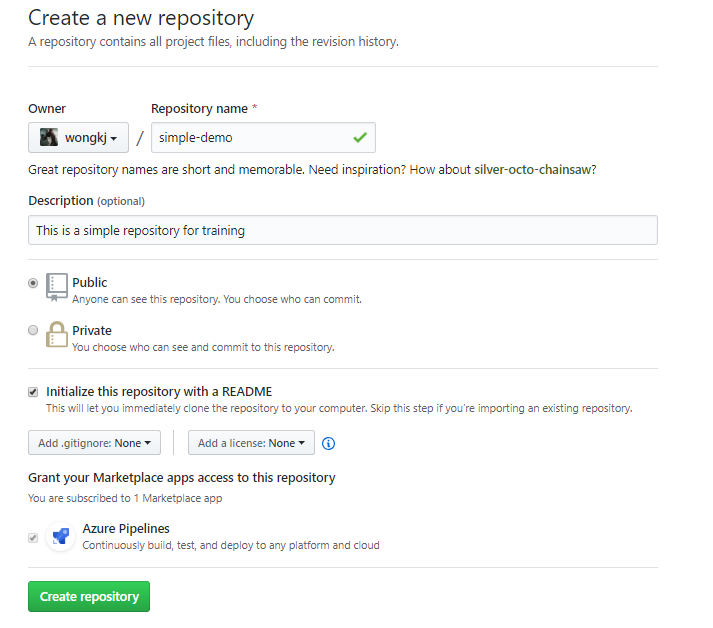


**LECTURE 20 – Projects**

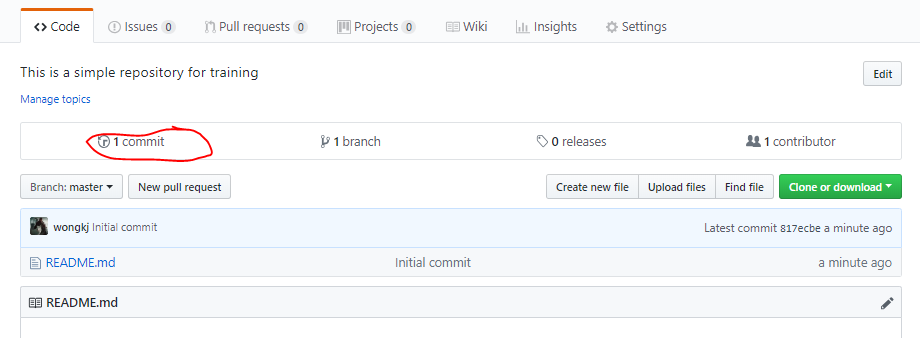
* We are now going to set up our Projects folder.
* In the **git-for-windows** folder create another folder called **projects**.

**LECTURE 23 – GitHub Repository**

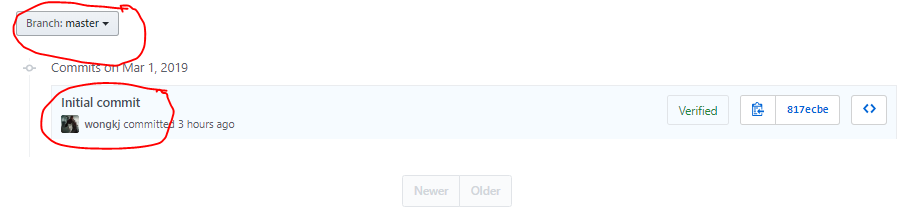
* Create a new Repository with the following details.



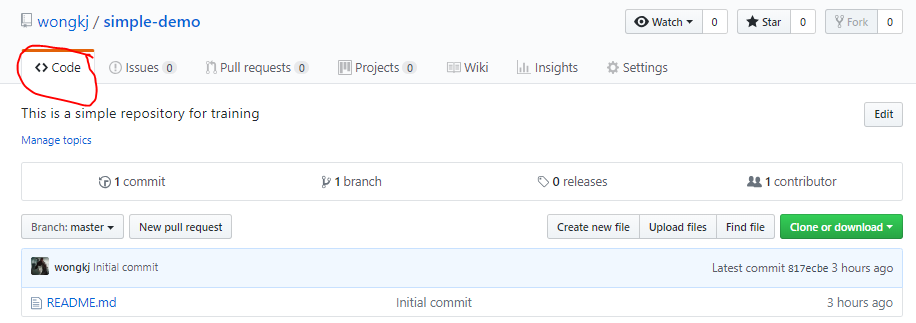
* To see the commits of the repo, click on the **commit** tab.



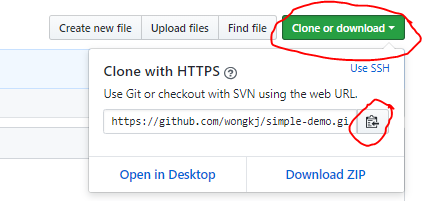
* On our **master** branch here is the initial commit.



* Click on **code** to go back to the main page of the repository.



* We are now going to clone this repository.
* Click on the following and click on the clipboard to copy the URL of this repository.

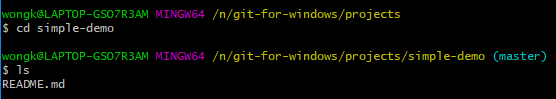


**LECTURE 24 - Clone**

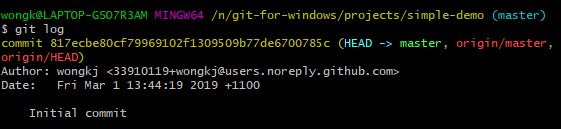
* Open **Git Bash** in the **git-for-windows/projects** folder.
* Type the following in the Git Bash terminal.
* **git clone <Repository URL>**
* **git clone <Repository URL> <new name of repository>** - This option will enable you to rename the repository to a new name instead of the default name of the repository.



* In Git Bash navigate into the **simple-demo** folder.

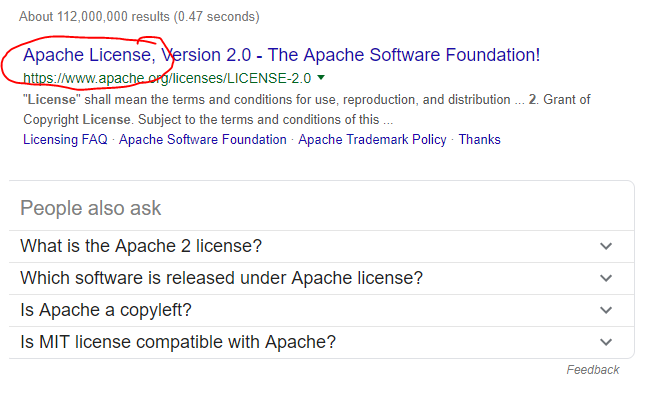


* Use **git status** to get the current state of the repository.
* To get the **commit history** of the repository, type the following in the terminal.
* **git log**

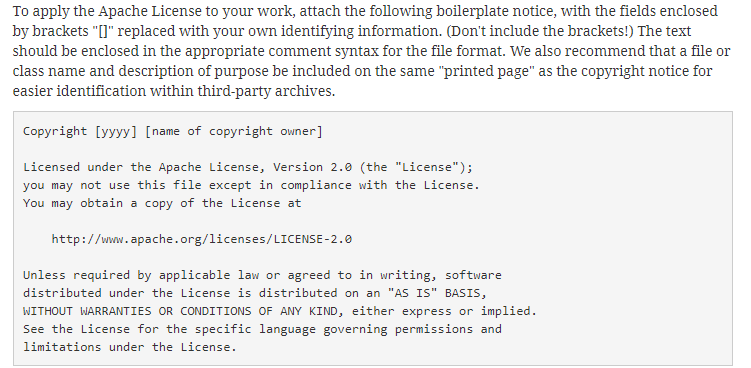


**LECTURE 25 – First Local Commit**

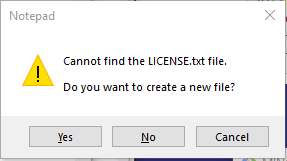
* The initial commit of our repository occurred when we initialized the repository on Git Hub with the README.md file.
* We are now going to add another file in to the **simple-demo** repository that we created.
* The file we are going to add will be the Apache 2.0 license.
* In the web browser, search for **Apache 2 license**.
* Click on the following link.



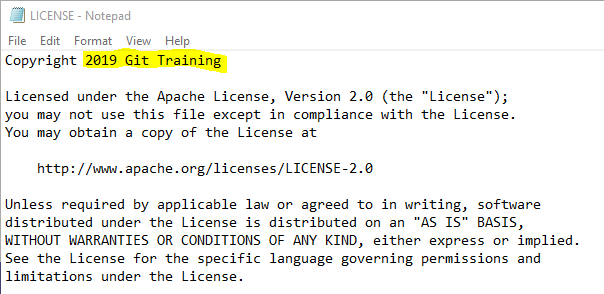
* Scroll all the way to the bottom and copy the following text.



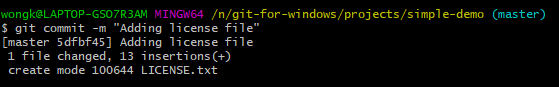
* Now we will create a text file and edit it through **notepad.**
* Type the following in the Git Bash terminal.
* **notepad LICENSE.txt.**
* Notepad will say they can’t find the text file and then will create the file. Hit **Yes**.



* Paste the text from before into the text file and change the following line.



* We will now commit the changes.
* To add the changes to tracking use the following.
* **git add <Name of file to add>** - Commit changes for specific file.
* **git add** – Commit all changes.
* To commit the changes use the following.
* **git commit -m <Commit Message>**

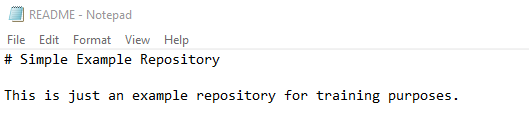


**LECTURE 26 – Edit Files**

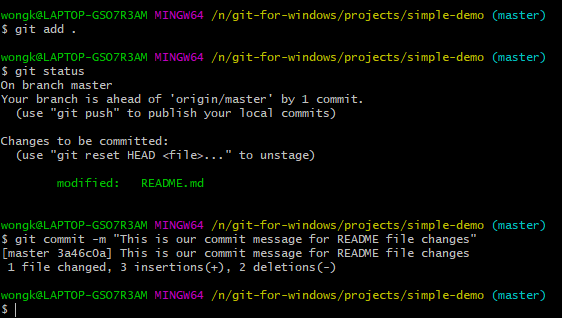
* Open the README.md file and change the text to the following.

# Simple Example Repository

This is just an example repository for training purposes.



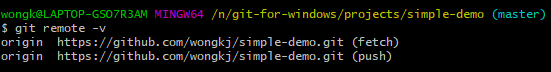
* Add and commit the changes with the following message “**This is our commit message for README file changes**”.



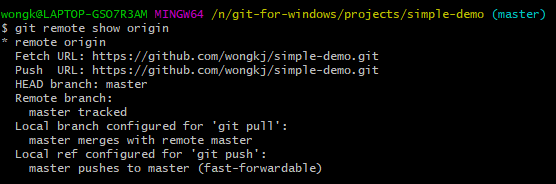
* If you just type **git commit** with no message you will be taken to **vi** where you will be able to write a message that’s multiple lines long”.

**LECTURE 27 – Push Back to GitHub**

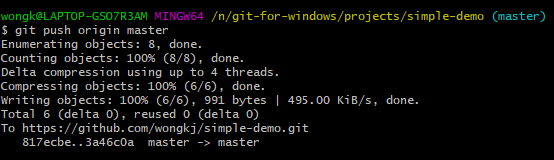
* When you clone a remote repository from GitHub onto your local machine, it automatically creates a reference back to that remote GitHub repository.
* To work with the configurations of the git remote repositories, type **git remote** in the Git Bash terminal.
* In the Git Bash terminal type the following to get the list of the configured git remote repositories.
* **git remote -v**



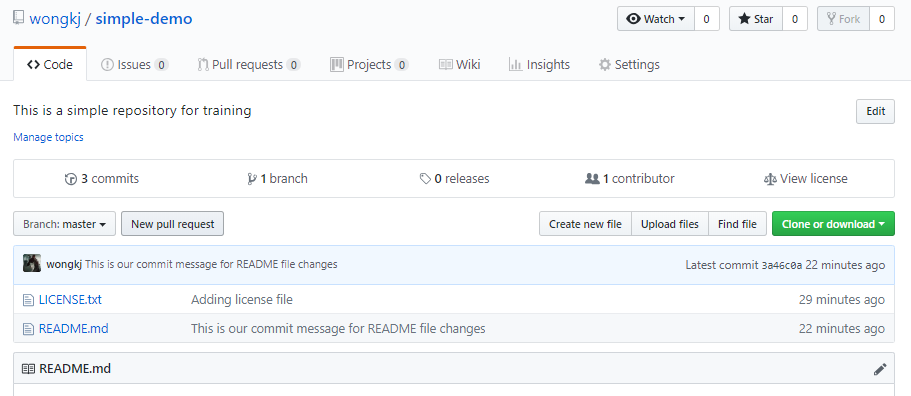
* As you can see above the name of the remote repository is **origin**.
* To find out more about this origin remote repository, type the following in the Git Bash terminal.
* **git remote show <name of remote repository>**



* Now in order to push our local changes to our remote repository on Git Hub we need 2 pieces of information.
* First, is the name of our remote repository, which in our case is **origin**.
* Second, is the name of the branch we are trying to push to the remote repository, which in our case is the **master** branch.
* Type the following in the Git Bash terminal to push the local changes in the **master** branch to the **origin** remote repository.
* **git push <name of remote repository> <name of local branch to push to remote repository>**



* Now when you go to the GitHub **simple-demo** remote repository and refresh the page, you should now see 3 commits and 2 files in the repository.



**LECTURE 29 – Command Prompt**

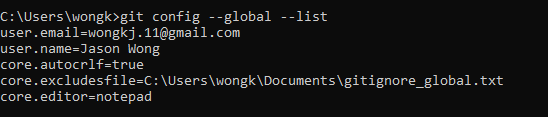
* We are now going to use Git through the windows command prompt.

**LECTURE 30 - Notepad**

* To set the main editor in git to notepad, type the following for the config.
* **git config –-global core.editor <name of editor you want as the core editor>**



* To see this change type the following in the command prompt.
* **git config –-global --list**

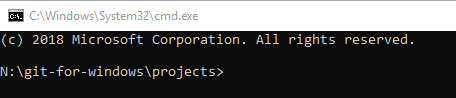


* To open notepad with our global settings file, type the following in the command prompt.
* **git config –-global -e**



**LECTURE 31 – Start Locally**

* What we are going to do now is create a repository locally and then pushing that to a remote repository.
* Navigate to the **projects** folder in our command prompt.



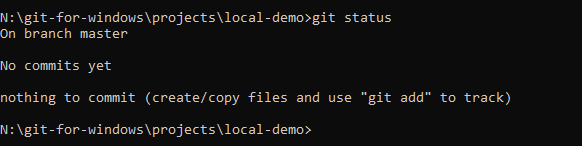
* We are going to initialize a repository in to our projects folder.
* In the terminal type.
* **git init <Name of repository>**



* In the command prompt navigate into the **local-demo** folder.



* Type **git status** to get the current status of changes and commits.



**LECTURE 32 – First Commit**

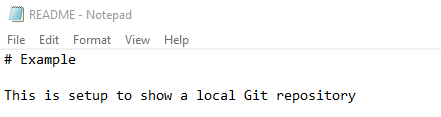
* We are first going to create the README.md file.
* In the command prompt type.
* **notepad README.md**



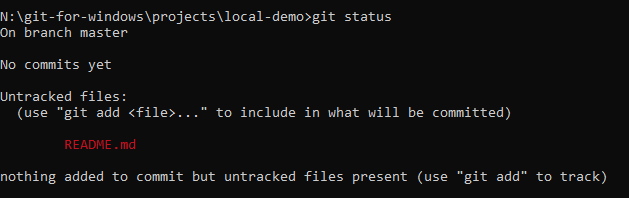
* Type the following in the README.md file.

# Example

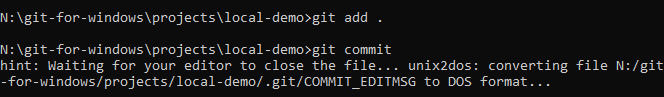
This is setup to show a local Git repository



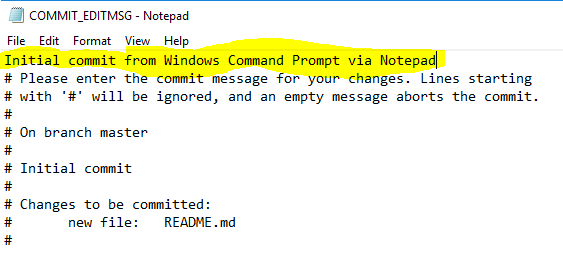
* Typing **git status** in the command prompt will display the following untracked files.



* Add all untracked changes.
* Then type **git commit** to open notepad for committing.



* In notepad type **Initial commit from Windows Command Prompt via Notepad** and then save and close Notepad.

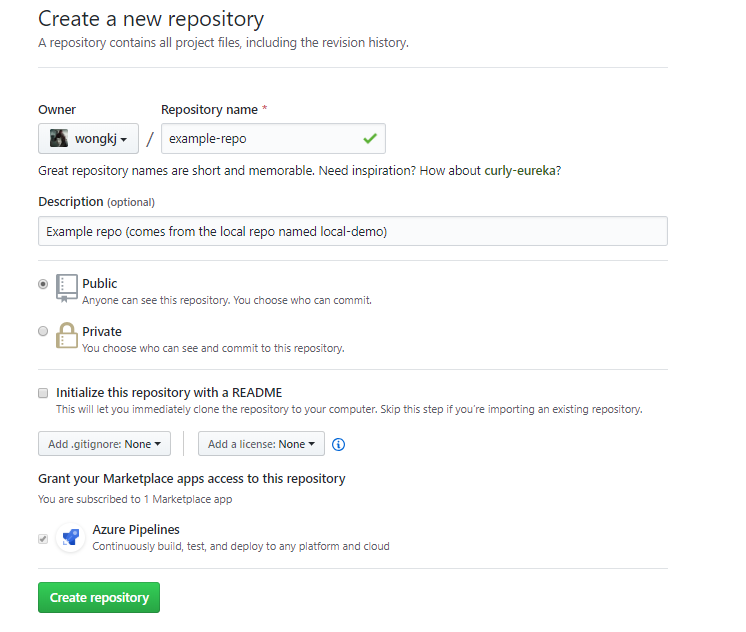


* That text we typed will be used as the commit message.

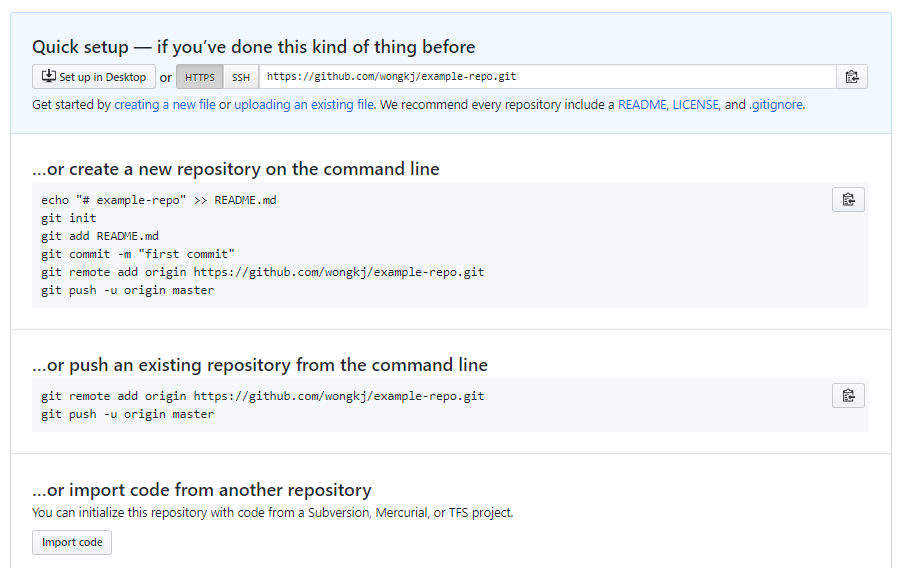
**LECTURE 33 – Going Remote**

* So far we have a local repository for the **local-demo** but we don’t have a remote repository for it yet.
* Go to **Git Hub** and create a new repository. Although you don’t have to name this remote repository the same as the local repository it’s probably best that you align the names. In this scenario, we will try giving this remote repository a different name to the local repository.

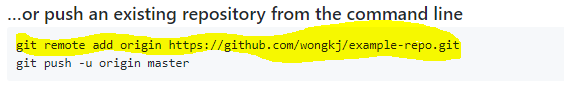
**Note:** Since we already have a README file in our local repository we don’t need to create one here.



* Since we are not initializing any files for our new repository we will get the following page giving instructions on how to link our remote repository locally.



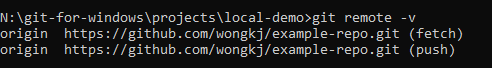
* Copy the section here.



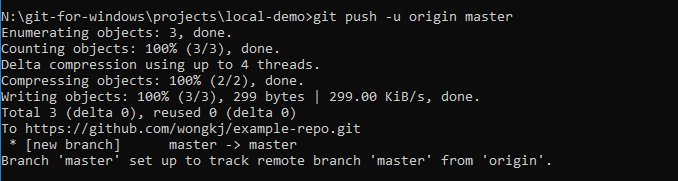
* Open the windows command prompt onto the local repository that you want to push to the remote repository.
* Paste the copied text in to the command prompt and press ENTER.



* To test that the **example-repo** remote repository has been connected to our **local-demo** local repository, type **git remote -v**.



* Now we’ll push our existing master branch to git hub.
* Type the following in the terminal.
* **git push -u origin master** – the **-u** sets up a tracking relationship between the local and remote repository on the master branch. Meaning the master branch locally will always be linked to the master branch on the remote side. We use this because as of now we don’t have a tracking relationship between the local and remote branch set up yet as we initialized this repository locally first instead of cloning an existing remote repository.



**LECTURE 34 - PowerShell**

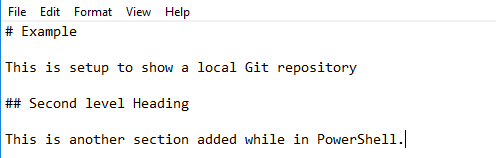
* Open the PowerShell in the **projects > local-demo** folder.
* Amend the README.md file to display the following.

# Example

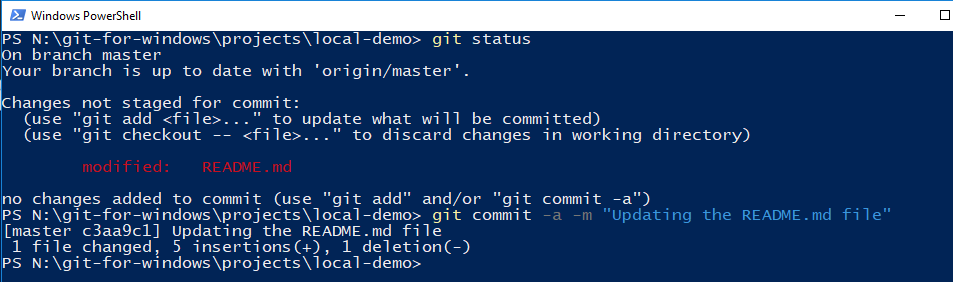
This is setup to show a local Git repository

## Second level Heading

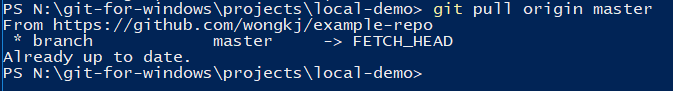
This is another section added while in PowerShell.



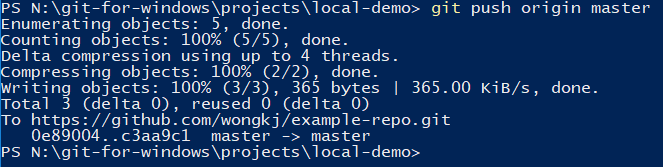
* To do an add and commit you can use **git commit -a -m <Commit message>.**
* In the PowerShell type the following: **git commit -a -m “Updating the README.md file”**.



* In a real-world team environment, before we push any of our commits done on our local repository onto our remote repository we will want to make sure that we download and integrate any changes done on the remote repository by other team members before we push any of our commits on to the remote repository and might potentially overwrite our team members commits.
* To pull and integrate any commits done on the remote repository to our local repository we type the following in to the terminal.
* **git pull origin master**

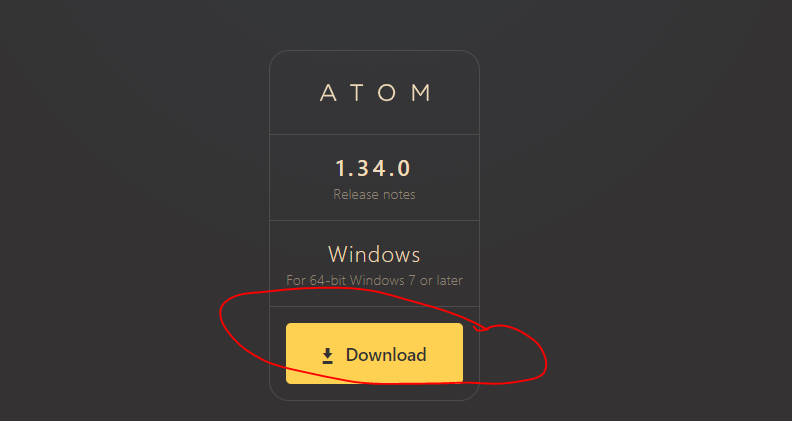


* **ALWAYS PULL BEFORE PUSHING ANY COMMITS**.
* Once you’ve done the Pull, Push your local commits to the remote repository.

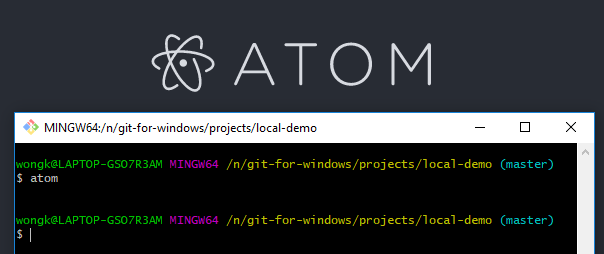


**LECTURE 36 – Atom Install**

* Navigate to [**https://atom.io**](https://atom.io)
* Click on **Download**.



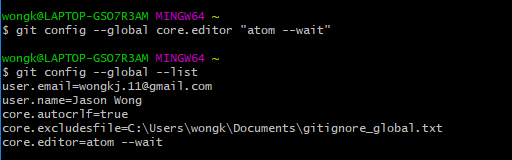
* If you want to invoke atom from the command line, open up Git Bash and just type **atom**. The atom text editor should pop up.



**LECTURE 37 – Atom Fonts**

**LECTURE 38 – Atom with Git**

* Open Git Bash.
* We are now going to configure Git so that it associates Atom as the core editor.
* In Git Bash type.
* **git config –-global core.editor “atom –-wait”**
* Use **git config –-global –-list** to get a list of all the config variables.

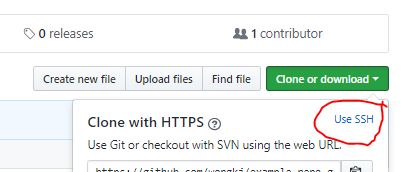


**LECTURE 39 – Using Atom**

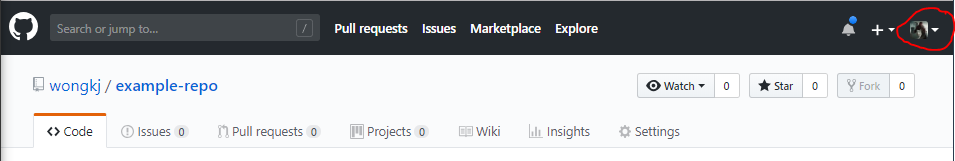
**LECTURE 40 – Atom and Windows**

**LECTURE 42 – Setup SSH**

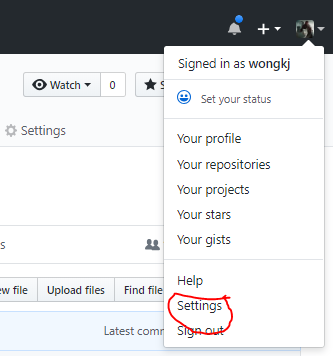
* **SSH** (Secure Shell) is a cryptographic network protocol for operating network services securely over an unsecured network.
* Go to the Git Hub Repository and navigate to **wongkj/example-repo**
* Click on **Clone or download** and then click on **Use SSH**.



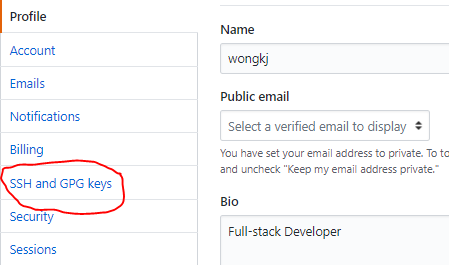
* We are first going to establish our key on our local system.
* In the repository front page click on your **Profile** picture.



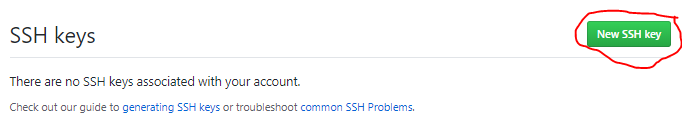
* Click on **Settings.**



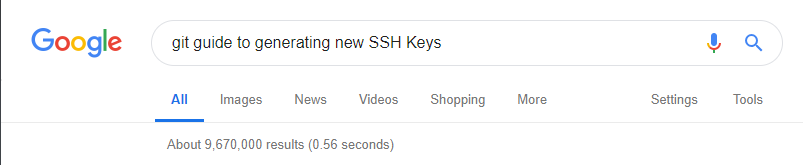
* Click on **SSH and GPG Keys**.



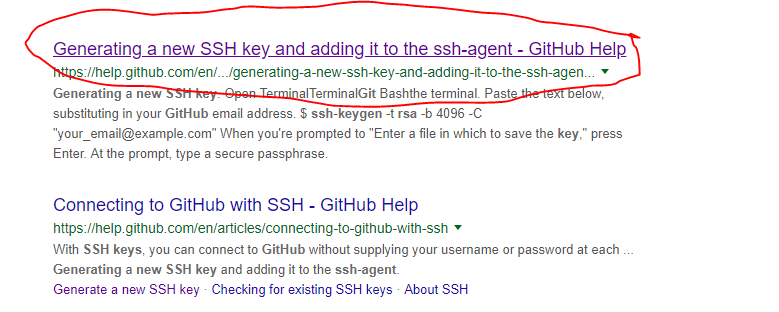
* Click on **New SSH Key**.



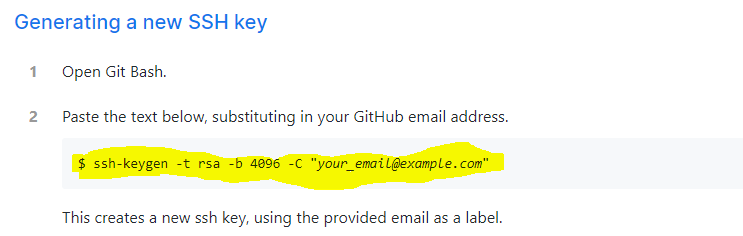
* To get help on how to generate a new SSH Key, go to a web browser and search for **git guide to generating new SSH Keys**.



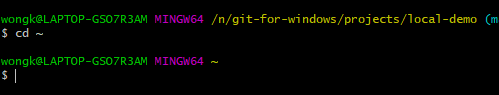
* Click on the following link.



* Copy the following string of text.



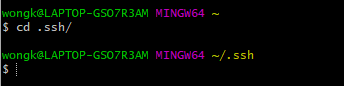
* Launch Git Bash.
* First, we’ll check that there’s no SSH Keys installed already.
* Navigate to the User’s home directory by typing **cd ~**.



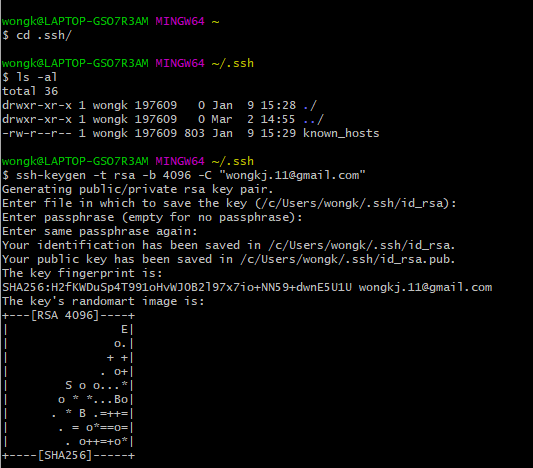
* We need to search for an **ssh** folder.
* Type **ls -al** to list all files and folders in the home directory.
* Look for a **.ssh** folder.



* If one doesn’t exist then create one using **mkdir .ssh**.
* Otherwise, navigate into the folder by typing **cd .ssh/**



* We are now going to create our ssh key by pasting in the command that we copied from the web site.
* Press Enter and keep pressing Enter until the SSH Key has been created.



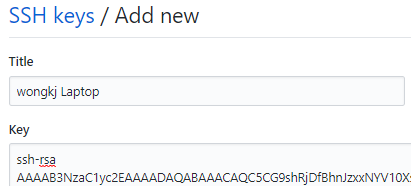
* Type **ls -al** to list all the files in the .ssh folder, including the SSH Key files.



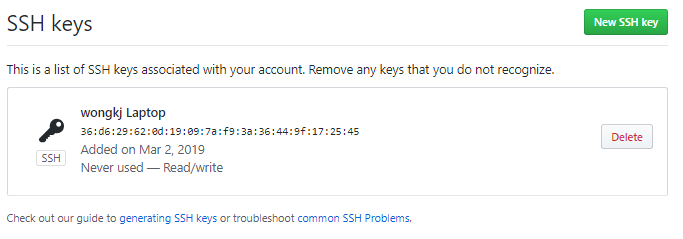
* You’ll notice there are two files called **id\_rsa** and **id\_rsa.pub**. These are the 2 SSH Key files and the **id\_rsa.pub** file is the public SSH Key.
* Open the **id\_rsa.pub** file by typing:
* **atom id\_rsa.pub**



* Copy the entire contents of this file.
* Go back to Git Hub and paste the contents in the **Key** section of the SSH Key creation page.

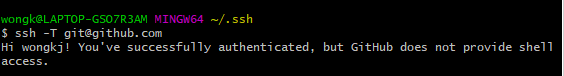


* Click on **Add SSH Key** and you will then be prompted to type in your password. Do that and click on **Confirm password**.
* Your SSH Key has now been added.



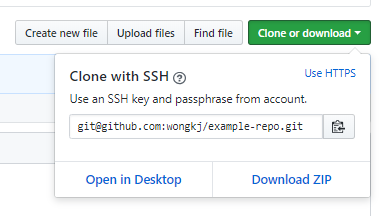
* We are now going to test the SSH Key connection.
* In Git Bash type the following.
* **ssh -T** [**git@github.com**](mailto:git@github.com)



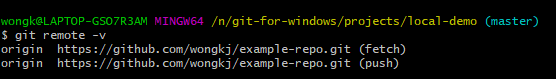


**LECTURE 43 – Working with Remotes**

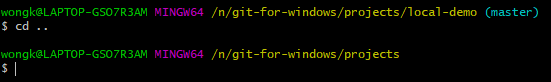
* Go to GitHub and go to the **example-repo** repository.
* Copy the **SSH** **URL** of the repo.



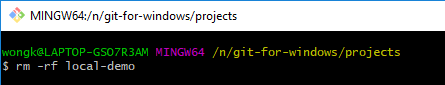
* Open Git Bash in the **local-demo** repository.
* Use **git remote -v** to see what the associated remote repository is for the **local-demo** repo.
* As you can see it is the **example-repo** remote repository.



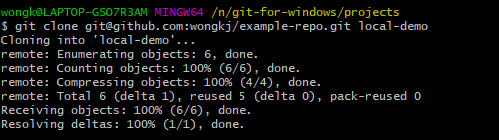
* We are going to delete the **local-demo** repository.
* First, ensure all changes are committed in the repository.
* In Git Bash navigate one directory out into the **projects** directory.



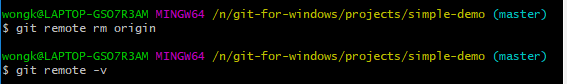
* Type the following to remove the directory.
* **rm -rf local-demo**



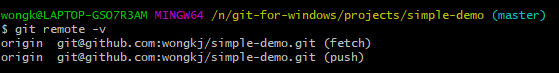
* We will now use the **git clone** command to clone the remote repository once again.
* Type the following to cloned the repository locally.
* **git clone <Git Remote Repository SSH URL> <Optional – alternative name of new local repository>**



* With Git Bash, navigate to the **simple-demo** repository.
* We will now change the remote repository tracking connection from the HTTPS version to the SSH version.
* Type the following to destroy the remote repository tracking connection.
* **git remote rm origin**

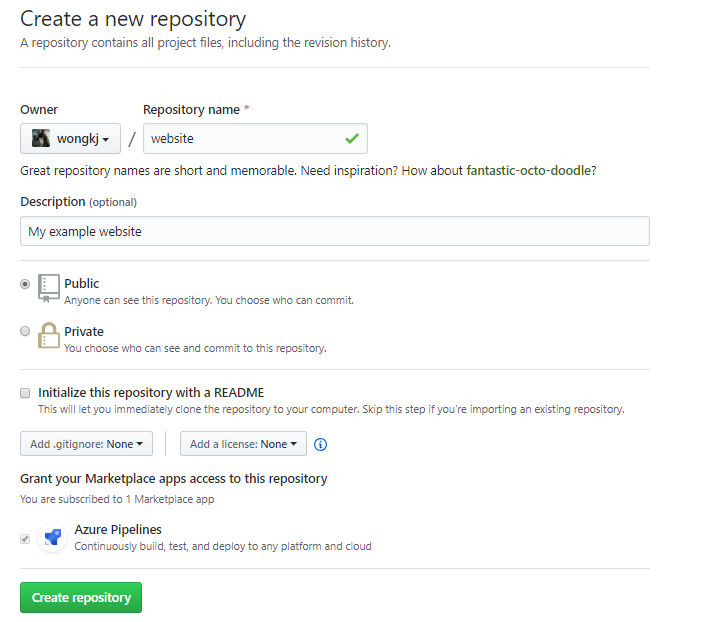


* Copy the GitHub remote simple-demo repository URL.
* Type the following in the terminal to re-establish the tracking connection between our local repository and our remote repository but this time we use the SSH URL instead of the HTTPS URL.
* **git remote add origin <SSH URL>**

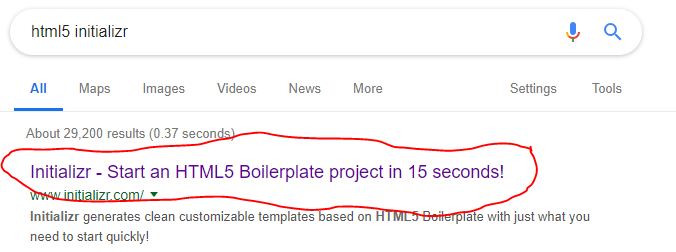


**LECTURE 44 – Start Local with an Existing Project**

* Here we are going to create a new Repository from some existing content.
* Create a new repository with the following details and it should have nothing initialized.



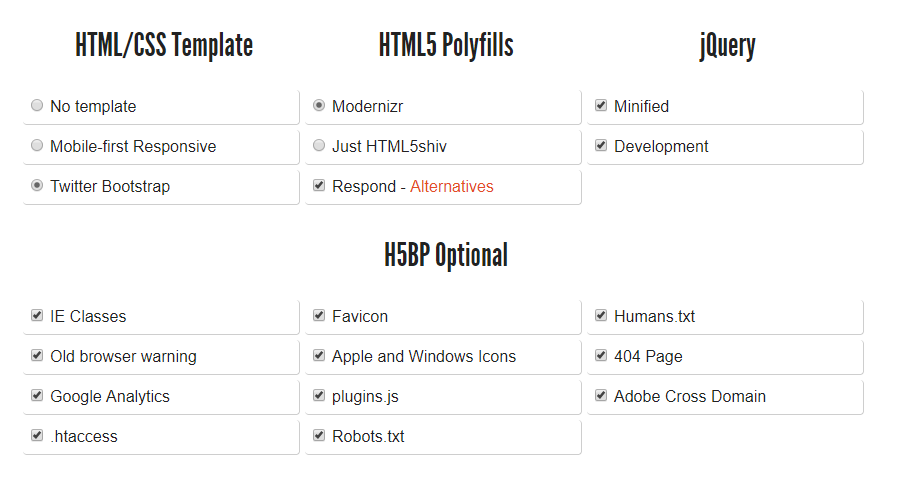
* We are now going to build a quick HTML application.
* In a web browser search for **HTML5 Initializr.**
* Click on the following link.



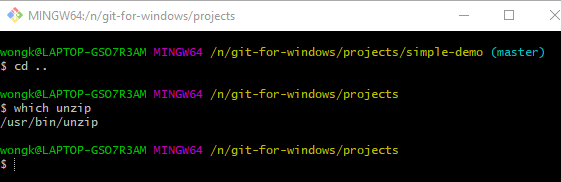
* Choose **bootstrap**.



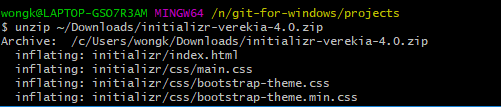
* Choose the following check-ins and then click on **Download it!**



* Go back to Git Bash and navigate to the **projects** folder.
* Type in **which unzip** to access the unzip functionality.
* **which unzip**



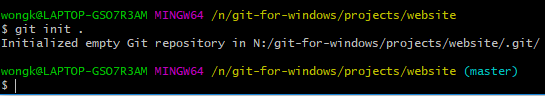
* Type in the location and name of the zip file you want to unzip.
* **unzip ~/Downloads/initializr-verekia-4.0.zip**



* The contents of the zipped file will be placed in a folder called initializr in the projects directory.
* We want to rename this folder from initializr to **website**.
* In Git Bash type the following.
* **mv initializr website**

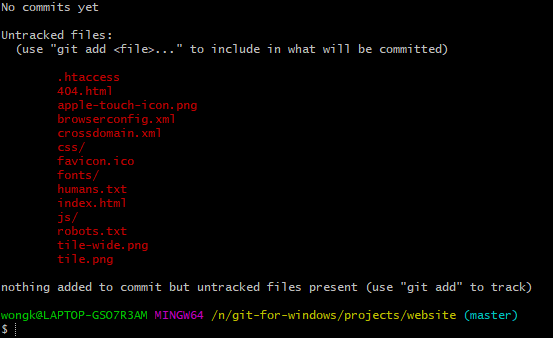


* In Git Bash, navigate into the **website** folder.
* We are now going to initialize this directory as a repository.
* In Git Bash type the following.
* **git init .**

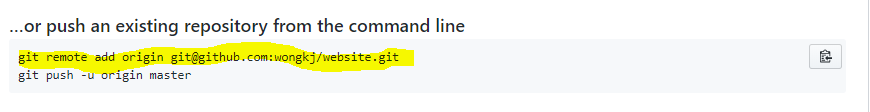


**LECTURE 45 – Recursive Add**

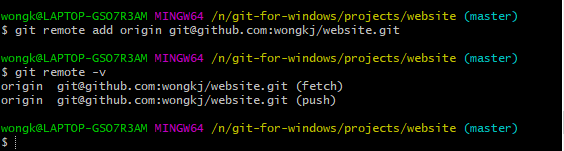
* In Git Bash in the **website** repository, type **git status.**



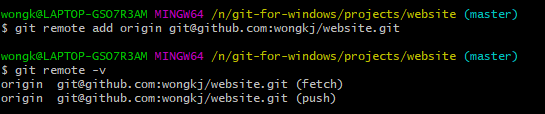
* Type **git add .** to add all the files for staging.
* Do an initial commit with the message **“Initial Commit”**.
* We will now link the locally created repository with the Git Hub remote repository.
* In Git Hub go to the **website** repository and copy the following.



* In Git Bash, paste the string of text that we just copied. This will establish the connection.



* Type the following. The **-u** is used to establish the tracking relationship between the local repository and the remote repository.
* **git push -u origin master**



* Go to Git Hub to the **website** repository.
* Refresh the page, and you should see all the files from the local repository now in the remote repository.

**LECTURE 46 - Delete**

* Here we are going to learn how to delete unwanted files inside of Git and outside of Git.
* In Git Bash, navigate to the following folder.

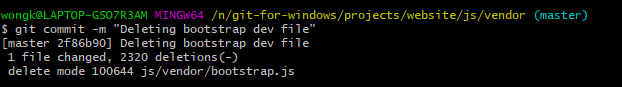
**~** **projects\website\js\vendor**



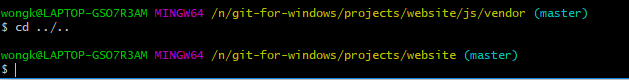
* We are going to delete the **bootstrap.js** file.
* In Git Bash type the following.
* **git rm bootstrap.js**



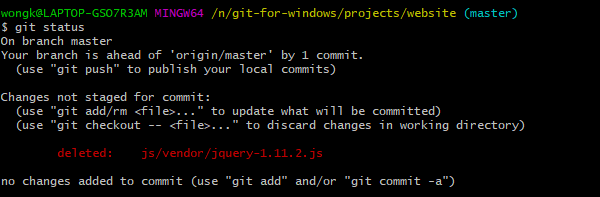
* Commit the deletion with the message **“Deleting boostrap dev file”**.



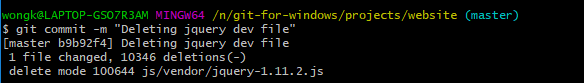
* In Git Bash, navigate back to the **~projects/website** folder.



* In Windows Explorer navigate to the **~projects/website/js/vendor** folder and delete the **jquery-1.11.2.js** file manually.
* In Git Bash, type **git status**.



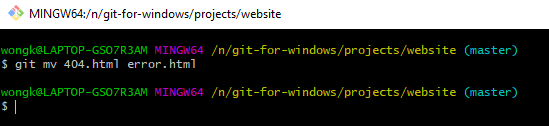
* Add the changes to staging and then commit the changes with the message **“Deleting jquery dev file”**



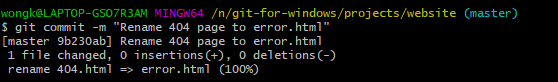
* Now we will remove the **plugins.js** file by typing.
* **rm plugins.js**
* Commit the changes with the message **“Delete plugins js file”**.

**LECTURE 47 - Rename**

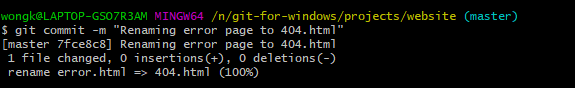
* We are going to rename a file.
* Go to Git Bash and navigate to the **website** directory.
* In the terminal type.
* **git mv <name of file to rename> <new name of file>**



* Commit the changes with the message **“Rename 404 page to error.html”**

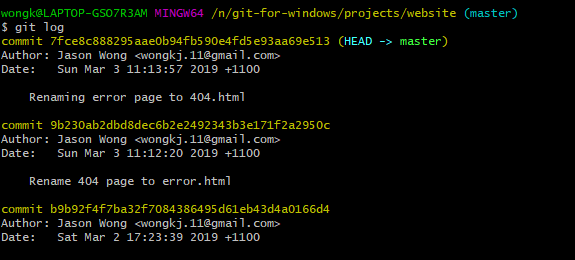


* Through the windows explorer, rename the error.html file to 404.html.
* Go back to Git Bash.
* Add and commit the change with the message **“Renaming error page to 404.html”**.

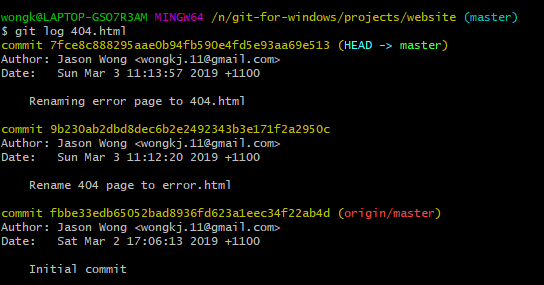


**LECTURE 48 – Git Log**

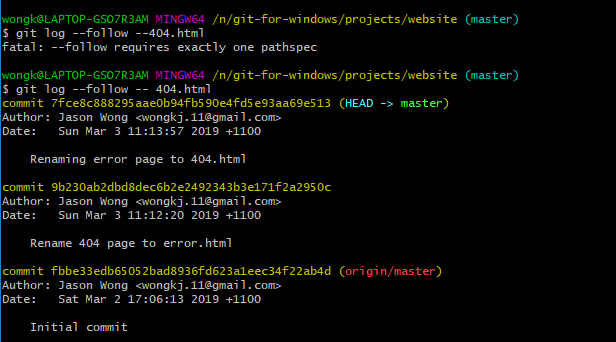
* In the Git Bash terminal, type **git log**.



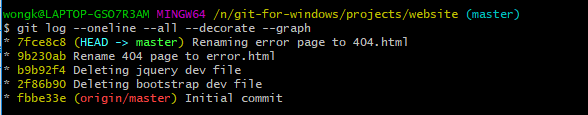
* We will now look into seeing all the commits for a specific file.
* The syntax for listing all the commits of a specific file, type:
* **git log <Name of file>**



* To get a list of all the changes of a file including any renamings, use the following syntax.
* **git log –-follow -- <Name of file>**



* We will now use a special set of git log options to list all the commits in a one-line decorated set of strings.
* The syntax for this special command is as follows.
* **git log –-oneline –-all –-decorate –-graph**



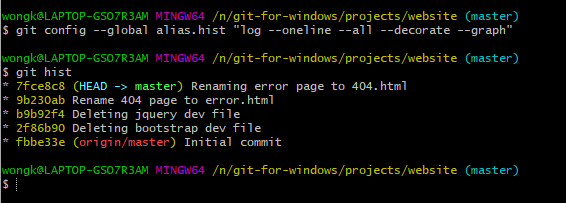
* Here is what each option stands for.
* **--oneline:** This will consolidate all our commit information into a single line.
* **--all:** Will show all the commits.
* **--decorate:** Will add all the decorations such as the name of the references (master branch, etc.)
* **--graph:** Will show the log items in a graph format so later on we can see the main line and all branches branching out from it.

**LECTURE 49 - Aliases**

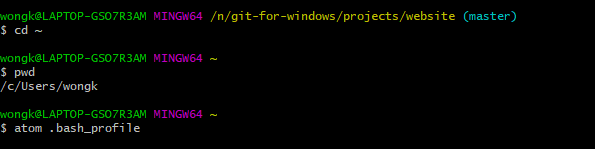
* **Aliases** are essentially stored procedures where we are able to save on to the config a string of options as a special command.
* Below we are saving a special command called **hist** into the config.
* **git config –-global alias.hist “log –-oneline –-all –-decorate –-graph”**



* In the terminal type.
* **git hist**



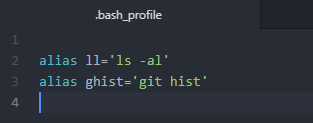
* we are now going to create our own **bash commands**.
* In Git Bash navigate to the root folder by typing **cd ~**.
* We need to create a new file called **.bash\_profile** file. We will do this using Atom.
* In the Git Bash terminal type **atom .bash\_profile**. Atom will then open up with a new file called **.bash\_profile.**



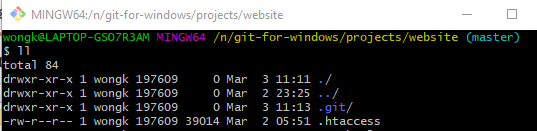
* In atom create a new alias called **ll** that will list all files in a directory.
* Also, create another alias that will combine **git** and **hist**.

alias ll='ls -al'

alias ghist='git hist'



* Close and Open Git Bash.
* Type **ll** to confirm that the ll command works.



**LECTURE 50 – Ignore Unwanted Files**

* Open Git Bash in the **website** folder.
* Use atom to create a file called **app.log**.
* **atom app.log**



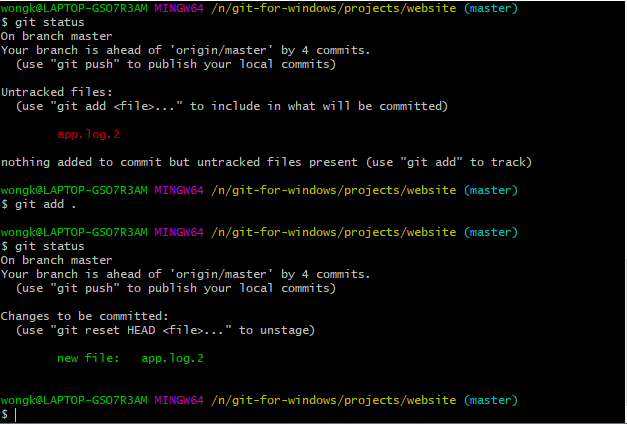
* Write a whole bunch of random text in the app.log file and then save and close atom.

**LECTURE 51 – Backout Changes**

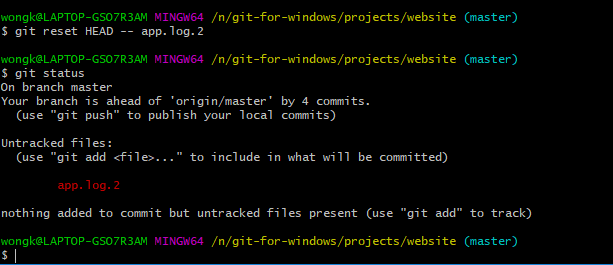
* We are going to learn how to back out of pre-commit changes.
* We’ll start by renaming **app-2.log** to **app.log.2.**
* **mv app.log app.log.2**



* Add this change to tracking by using **git add .**



* Use **git reset** to reset all stagings.
* In the terminal type.
* **git reset HEAD -- <name of file we want to reset changes on>.**

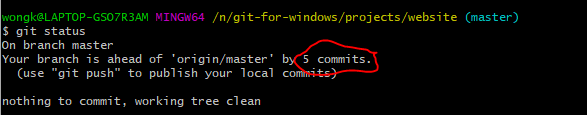


* To revert a file back to the last version before the current changes you use the following syntax.
* **git checkout -- <name of file to revert>**

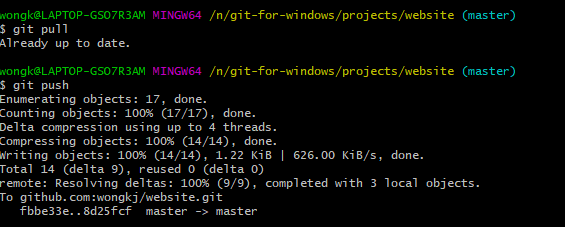


**LECTURE 51 – Going Remote to GitHub**

* In the terminal type **git status**.
* As you can see below, our local repository is 5 commits ahead of our remote repository.

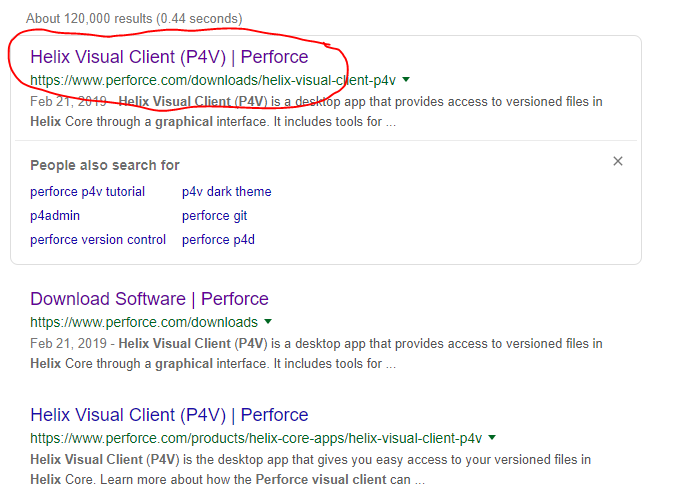


* Use **git pull.**
* Then use **git push.**

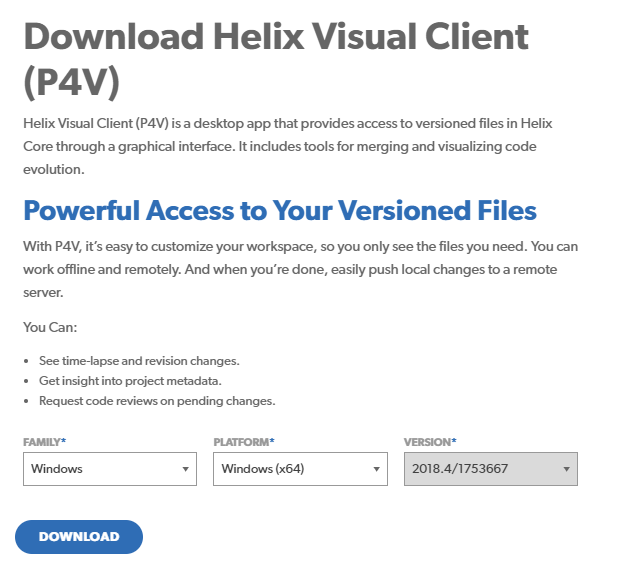


**LECTURE 54 – Install Diff and Merge Tool**

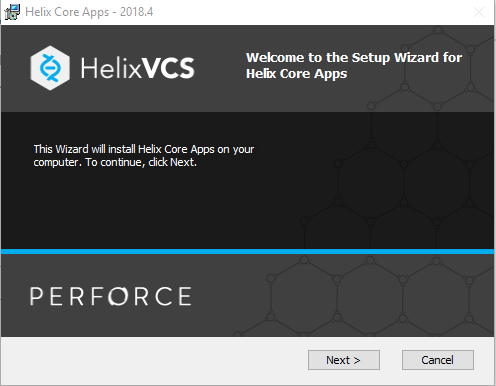
* We will download some tools relating to Differences and Merging.
* Go to a web browser and search for **helix p4v visual client**.
* Click on the following link.



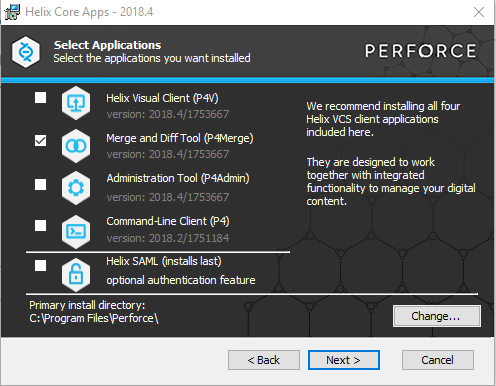
* Select the following options then click on **Download**.



* Once the executable has finished downloading open it to run the installation.
* Click on **Next.**



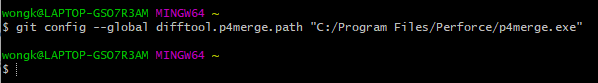
* Uncheck all the boxes except the **Merge and Diff Tool** then click on **Next**.



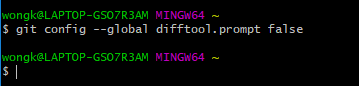
* Click on **Install** to finish the installation process.
* Now we need to merge p4merge with Git.
* First locate P4 Merge on our system. It should be located at **C:\Program Files\Perforce\p4merge.exe**
* Now start up Git Bash.
* We will configure the Git Diff Tool to use p4merge for doing diffs.
* First, we will name our diff tool to p4merge by using the following.
* **git config –-global diff.tool p4merge.**



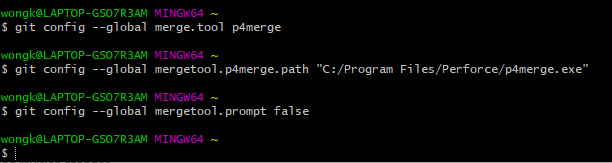
* We now need to set the path of p4merge.
* Type the following.
* **git config –-global difftool.p4merge.path “C:/Program Files/Perforce/p4merge.exe”**



* Now type the following **git config –-global difftool.prompt false.**



* We will now configure our merge tool.
* The mergetool is similar to the difftool except you replace diff with merge.



**LECTURE 55 – The Setup**

* In this lecture we are going to set up a repository to test our comparisons and merges.
* In Git Bash, navigate to the **website** repository.
* We are going to modify the **index.html** file.
* Open the index.html file in atom by typing **atom index.html**.



* Here we are going to make changes and save them incrementally so that we can do our comparisons.
* Make the following changes and then save the file.

<!doctype html>

<!--[if lt IE 7]> <html class="no-js lt-ie9 lt-ie8 lt-ie7" lang=""> <![endif]-->

<!--[if IE 7]> <html class="no-js lt-ie9 lt-ie8" lang=""> <![endif]-->

<!--[if IE 8]> <html class="no-js lt-ie9" lang=""> <![endif]-->

<!--[if gt IE 8]><!--> <html class="no-js" lang=""> <!--<![endif]-->

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<title>Remote Example</title>

<meta name="description" content="">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="apple-touch-icon" href="apple-touch-icon.png">

<link rel="stylesheet" href="css/bootstrap.min.css">

<style>

body {

padding-top: 50px;

padding-bottom: 20px;

}

</style>

<link rel="stylesheet" href="css/bootstrap-theme.min.css">

<link rel="stylesheet" href="css/main.css">

<script src="js/vendor/modernizr-2.8.3-respond-1.4.2.min.js"></script>

</head>

<body>

<!--[if lt IE 8]>

<p class="browserupgrade">You are using an <strong>outdated</strong> browser. Please <a href="http://browsehappy.com/">upgrade your browser</a> to improve your experience.</p>

<![endif]-->

<nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

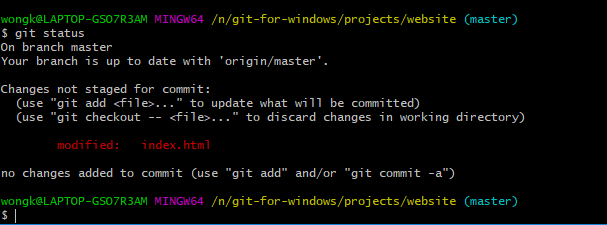
<span class="icon-bar"></span>

</button>

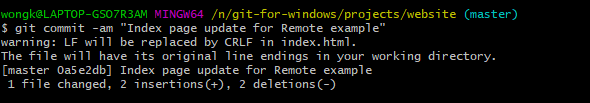
<a class="navbar-brand" href="#">Remote Example</a>

</div>

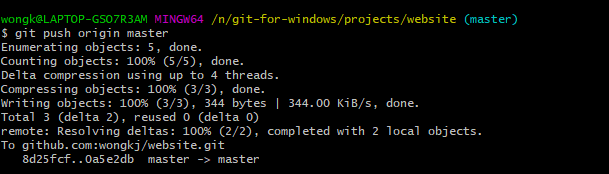
* If we do **git status** you will see that we have a modified file.



* Commit the changes by typing.
* **git commit -am “Index page update for Remote example”**



* Push the changes.



* Go back to **index.html** and make the following changes.

<!doctype html>

<!--[if lt IE 7]> <html class="no-js lt-ie9 lt-ie8 lt-ie7" lang=""> <![endif]-->

<!--[if IE 7]> <html class="no-js lt-ie9 lt-ie8" lang=""> <![endif]-->

<!--[if IE 8]> <html class="no-js lt-ie9" lang=""> <![endif]-->

<!--[if gt IE 8]><!--> <html class="no-js" lang=""> <!--<![endif]-->

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<title>Repository First Example</title>

<meta name="description" content="">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="apple-touch-icon" href="apple-touch-icon.png">

<link rel="stylesheet" href="css/bootstrap.min.css">

<style>

body {

padding-top: 50px;

padding-bottom: 20px;

}

</style>

<link rel="stylesheet" href="css/bootstrap-theme.min.css">

<link rel="stylesheet" href="css/main.css">

<script src="js/vendor/modernizr-2.8.3-respond-1.4.2.min.js"></script>

</head>

<body>

<!--[if lt IE 8]>

<p class="browserupgrade">You are using an <strong>outdated</strong> browser. Please <a href="http://browsehappy.com/">upgrade your browser</a> to improve your experience.</p>

<![endif]-->

<nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

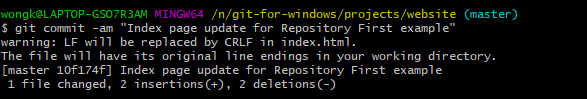
<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#">Repository First Example</a>

</div>

* Perform the following commit.
* **git commit -am “Index page update for Repository First example”**



* Go back to atom and make the following changes.

<!doctype html>

<!--[if lt IE 7]> <html class="no-js lt-ie9 lt-ie8 lt-ie7" lang=""> <![endif]-->

<!--[if IE 7]> <html class="no-js lt-ie9 lt-ie8" lang=""> <![endif]-->

<!--[if IE 8]> <html class="no-js lt-ie9" lang=""> <![endif]-->

<!--[if gt IE 8]><!--> <html class="no-js" lang=""> <!--<![endif]-->

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<title>Repository Second Example</title>

<meta name="description" content="">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="apple-touch-icon" href="apple-touch-icon.png">

<link rel="stylesheet" href="css/bootstrap.min.css">

<style>

body {

padding-top: 50px;

padding-bottom: 20px;

}

</style>

<link rel="stylesheet" href="css/bootstrap-theme.min.css">

<link rel="stylesheet" href="css/main.css">

<script src="js/vendor/modernizr-2.8.3-respond-1.4.2.min.js"></script>

</head>

<body>

<!--[if lt IE 8]>

<p class="browserupgrade">You are using an <strong>outdated</strong> browser. Please <a href="http://browsehappy.com/">upgrade your browser</a> to improve your experience.</p>

<![endif]-->

<nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

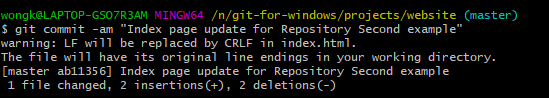
<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#">Repository Second Example</a>

</div>

* Make another commit with the following message.
* **git commit -am “Index page update for Repository Second example”**



* Make the last following changes.

<!doctype html>

<!--[if lt IE 7]> <html class="no-js lt-ie9 lt-ie8 lt-ie7" lang=""> <![endif]-->

<!--[if IE 7]> <html class="no-js lt-ie9 lt-ie8" lang=""> <![endif]-->

<!--[if IE 8]> <html class="no-js lt-ie9" lang=""> <![endif]-->

<!--[if gt IE 8]><!--> <html class="no-js" lang=""> <!--<![endif]-->

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<title>Repository Third Example</title>

<meta name="description" content="">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="apple-touch-icon" href="apple-touch-icon.png">

<link rel="stylesheet" href="css/bootstrap.min.css">

<style>

body {

padding-top: 50px;

padding-bottom: 20px;

}

</style>

<link rel="stylesheet" href="css/bootstrap-theme.min.css">

<link rel="stylesheet" href="css/main.css">

<script src="js/vendor/modernizr-2.8.3-respond-1.4.2.min.js"></script>

</head>

<body>

<!--[if lt IE 8]>

<p class="browserupgrade">You are using an <strong>outdated</strong> browser. Please <a href="http://browsehappy.com/">upgrade your browser</a> to improve your experience.</p>

<![endif]-->

<nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

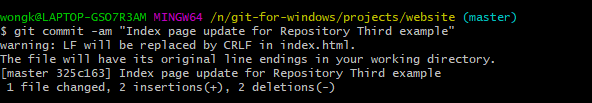
<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#">Repository Third Example</a>

</div>

* Make a commit with the following.
* **git commit -am “Index page update for the Third example”**



* Make the following changes.

<!doctype html>

<!--[if lt IE 7]> <html class="no-js lt-ie9 lt-ie8 lt-ie7" lang=""> <![endif]-->

<!--[if IE 7]> <html class="no-js lt-ie9 lt-ie8" lang=""> <![endif]-->

<!--[if IE 8]> <html class="no-js lt-ie9" lang=""> <![endif]-->

<!--[if gt IE 8]><!--> <html class="no-js" lang=""> <!--<![endif]-->

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<title>Stage Example</title>

<meta name="description" content="">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="apple-touch-icon" href="apple-touch-icon.png">

<link rel="stylesheet" href="css/bootstrap.min.css">

<style>

body {

padding-top: 50px;

padding-bottom: 20px;

}

</style>

<link rel="stylesheet" href="css/bootstrap-theme.min.css">

<link rel="stylesheet" href="css/main.css">

<script src="js/vendor/modernizr-2.8.3-respond-1.4.2.min.js"></script>

</head>

<body>

<!--[if lt IE 8]>

<p class="browserupgrade">You are using an <strong>outdated</strong> browser. Please <a href="http://browsehappy.com/">upgrade your browser</a> to improve your experience.</p>

<![endif]-->

<nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

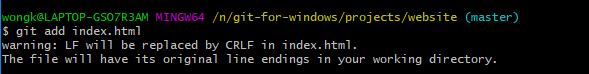
<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#">Stage Example</a>

</div>

* With these changes we just want to add them, so just type.
* **git add index.html**



* Make the following changes.

<!doctype html>

<!--[if lt IE 7]> <html class="no-js lt-ie9 lt-ie8 lt-ie7" lang=""> <![endif]-->

<!--[if IE 7]> <html class="no-js lt-ie9 lt-ie8" lang=""> <![endif]-->

<!--[if IE 8]> <html class="no-js lt-ie9" lang=""> <![endif]-->

<!--[if gt IE 8]><!--> <html class="no-js" lang=""> <!--<![endif]-->

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">

<title>Working Folder Example</title>

<meta name="description" content="">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="apple-touch-icon" href="apple-touch-icon.png">

<link rel="stylesheet" href="css/bootstrap.min.css">

<style>

body {

padding-top: 50px;

padding-bottom: 20px;

}

</style>

<link rel="stylesheet" href="css/bootstrap-theme.min.css">

<link rel="stylesheet" href="css/main.css">

<script src="js/vendor/modernizr-2.8.3-respond-1.4.2.min.js"></script>

</head>

<body>

<!--[if lt IE 8]>

<p class="browserupgrade">You are using an <strong>outdated</strong> browser. Please <a href="http://browsehappy.com/">upgrade your browser</a> to improve your experience.</p>

<![endif]-->

<nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">

<span class="sr-only">Toggle navigation</span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

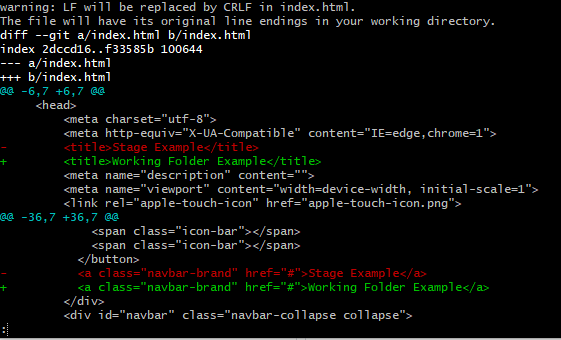
<a class="navbar-brand" href="#">Working Folder Example</a>

</div>

* We don’t want to add or commit these changes.

**LECTURE 56 – Comparing Differences**

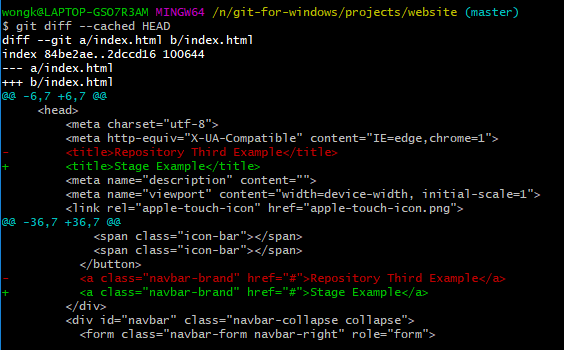
* We are now going to compare the **local states** in our Git Repository.
* The changes we are referring to will be the changes in our local directory versus staged changes versus what’s currently in our Git Repository.
* We are going to compare what’s in our **working directory** (What we’ve changed without staging) versus what’s currently in the **Git staging area**.
* In the terminal press type.
* **git diff**



* The changes in red are the ones that we have added to our **staging** area, while the ones in green are the ones that are currently located in our **working directory** but have not yet been staged.

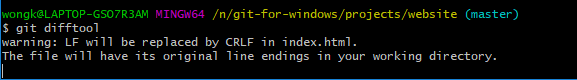
**NOTE:** HEAD is the last commit on our current branch.

* To compare the changes between the **staging** area and the last **commit** we use the following syntax.
* The word **–-cached** refers to the staging area.
* **git diff –-cached HEAD**

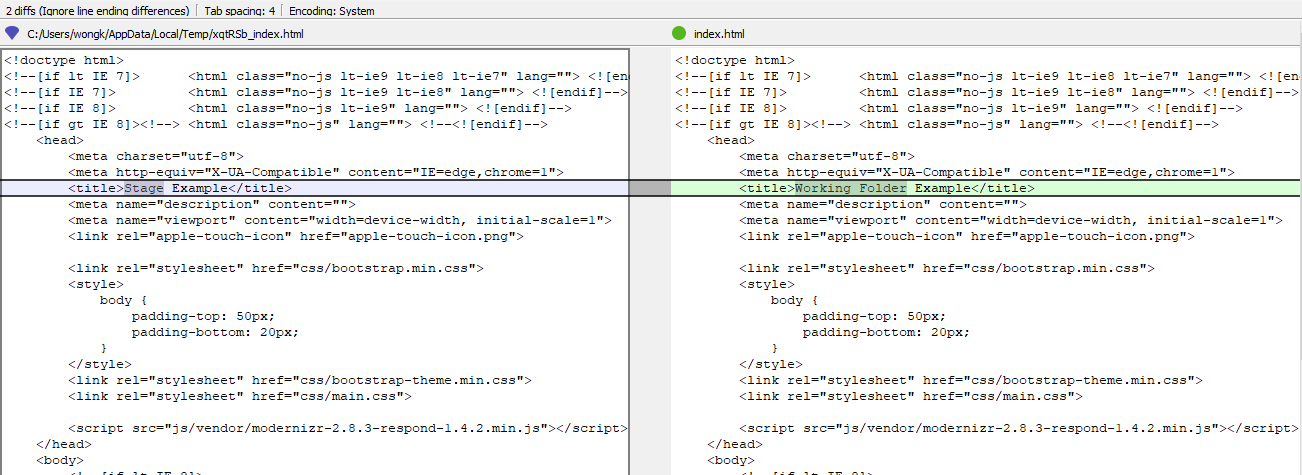


* We will now use the **p4merge** tool to see the differences between the **staged changes** and the last **commit**.
* In the terminal type the following to launch p4merge.
* **git difftool**

**Note:** difftool is the item we configured earlier.



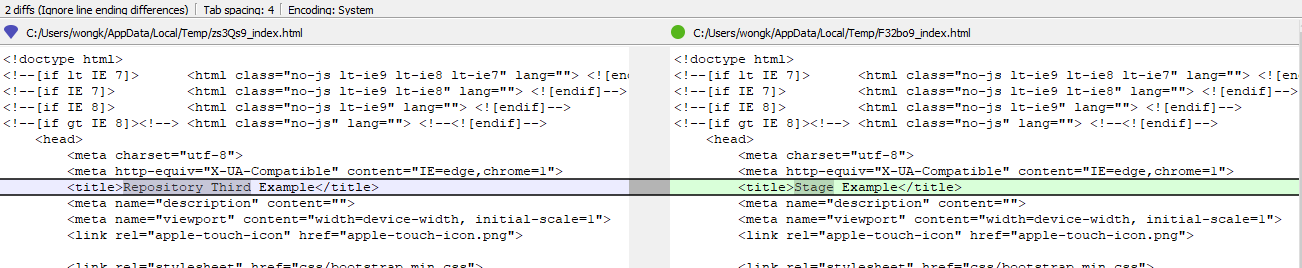
The left-hand side is the stage changes and the right-hand side is the commit changes.



* Exit **p4merge**.
* To see the changes between the current **working directory** and the last commit you just use.
* **git difftool –cached**

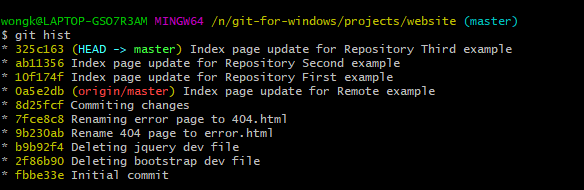


The changes on the left are the working directory changes, the ones on the right are the commit changes.

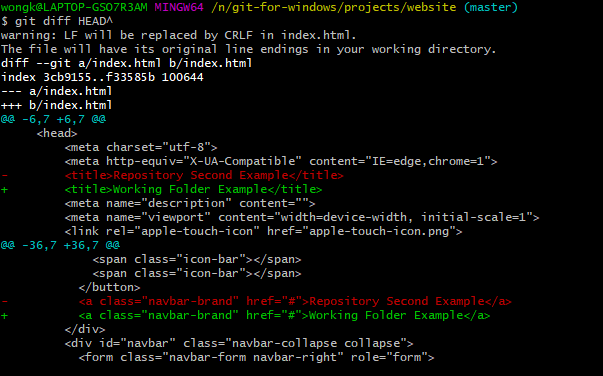


**LECTURE 57 – Compare Previous Changes**

* In the terminal type the custom function **git hist** to get a history of our commits.



* We will now compare the different commits against eachother.
* The following syntax will enable use to compare our last commit to the second-to-last commit (the commit right before our last commit).
* **git diff HEAD^**



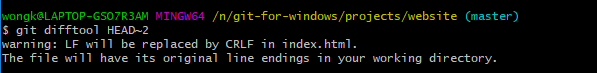
* We will now do the same thing using the **difftool**.
* Type the following in the terminal.
* **git difftool HEAD^**



* To go to the **third last** commit, use the following syntax.
* **git difftool HEAD^^**

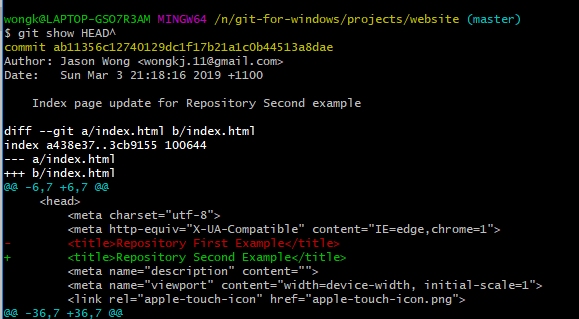


* Another way to write this is.
* **git difftool HEAD~2**

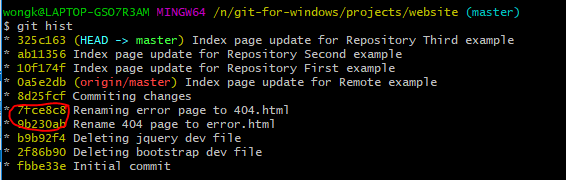


**LECTURE 58 – Git Show**

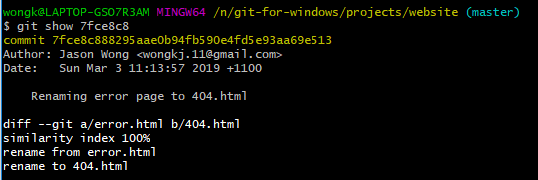
* We will now use the git **show** command to compare various different commit changes against the commit before it.
* Use the following syntax.
* **git show HEAD^**



* You can also use the commit Id to show the differences between the commits.
* First, get one of the commit id’s.

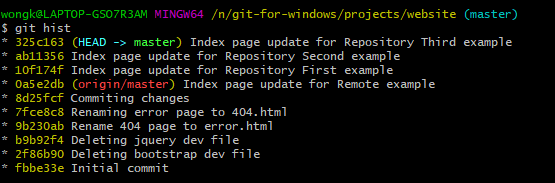


* Next, type the following syntax.
* **git show 7fce8c8**



**LECTURE 59 – Compare Previous Commits**

* We are now going to compare between 2 different commit points.
* We are now going to use **git difftool** to compare 2 different commits that are not necessarily close to eachother.
* Get the list of commits.



* In the terminal type.
* **git difftool 10f174f 9b230ab**

**Doesn’t seem to be working for me**

**LECTURE 60 – Compare Remote Changes**

* We are now going to compare our local stages and git repository versus the changes in the Remote Repository.

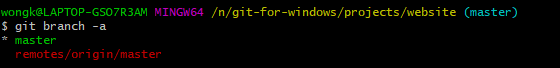
**Note:** With the **git pull** command, git will pull the latest changes in the Remote Git Repository and then merge them into our Local Repository.

**Note:** With the **git fetch** command, git will only pull the latest changes in the Remote Git Repository but not merge them.

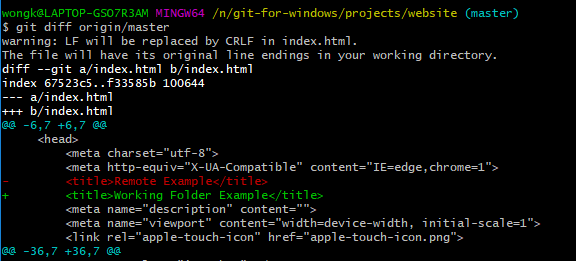
* In the terminal type the following.
* **git fetch –-all**



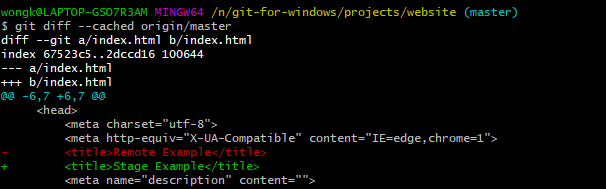
* By typing **git branch -a** you get a list of all the current branches. You’ll notice one called **remotes/origin/master**.
* This is the state of the remote repository that we fetched locally.



* We will now compare the **remotes/origin/master** branch to our **working folder**. Type the following.
* **git diff origin/master**

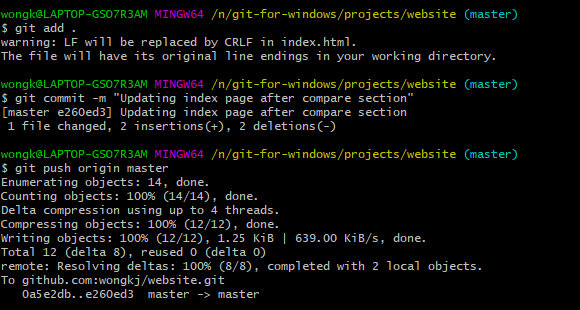


* We will now compare our cached (staging) state to the origin/master (remote) state.
* **git diff –-cached origin/master**



**LECTURE 61 – Just a bit of Cleanup**

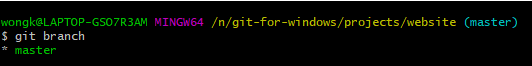
* Add and commit all the changes with the message **“Updating index page after compare section”**.



**LECTURE 63 – Easy Branching**

* In Git Bash navigate to the **website** project.
* We can create other branches aside from the master branch, make changes on those branches and then merge those branches with the master branch at a later point.
* To get a list of all the currently existing branches in a Repository, you just use the following syntax.
* **git branch**

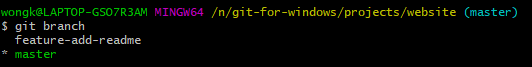
**NOTE:** The asterix indicates that the **master** branch is the current branch.



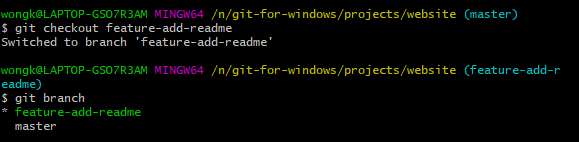
* We are going to create a new branch called **feature-add-readme**.
* Type the following.
* **git branch feature-add-readme**



* Although we are still on the **master** branch if you type **git branch** again you will notice that a new branch called **feature-add-readme** has been created.



* To change branches we need to use the **git checkout** command.
* In the terminal type the following to navigate to the **feature-add-readme** branch.
* **git checkout feature-add-readme**



* We will now use atom to create a **README.md** file in the repository.
* Type the following to add a README.md file using atom.
* **atom README.md**

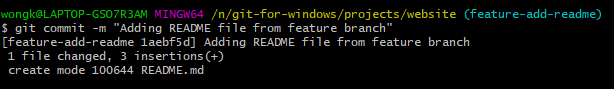


* In Atom, write the following text and then save the file, then close atom.

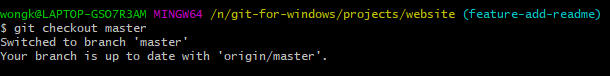
# ReadMe for Website Project

This is the readme file for this project

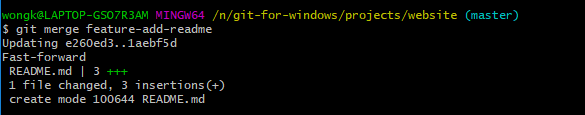
* Do a commit with the message “**Adding README file from feature branch”**.



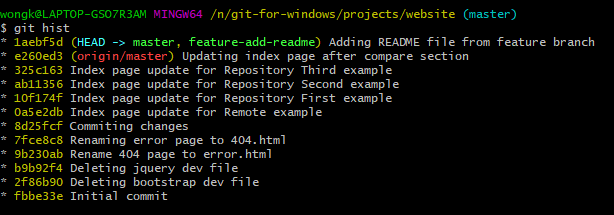
* Switch back to the master branch using git checkout.
* **git checkout master**



* To do the merge you need to type **git merge <Name of branch you want to merge into the current branch>**.
* Since we are currently in the master branch, we want to feature the changes from the **feature-add-readme** branch into our current **master** branch.
* Type the following to do that.
* **git merge feature-add-readme**

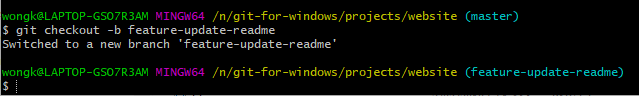


* If you type **git hist** you will see that the master and feature-add-readme branch now point to the same commit.



**LECTURE 64 – No Fast Forward Merges**

* We are now going to create another feature branch but this one is for updating our README file.
* We are going to combine **git branch** and **git checkout** into a single command to create a new branch and then navigating to that newly created branch.
* In the terminal type the following to create a new branch called **feature-update-readme**:
* **git checkout -b feature-update-readme**



* Use atom to update the README file.
* Type the following to open the README file in atom:
* **atom README.md**



* In atom, type the following, save the file and exit atom.

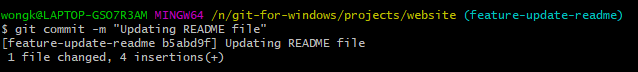
# ReadMe for Website Project

This is the readme file for this project

## Updates

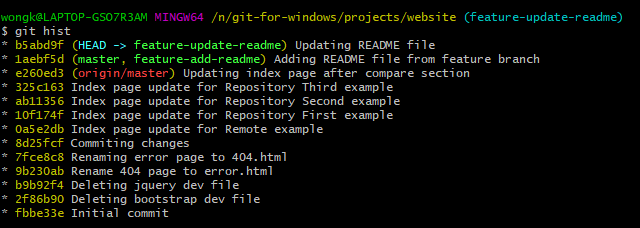
Adding changes for example

* In Git Bash do a commit with the message **“Updating README file”**.

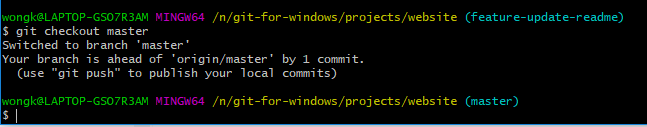


* In the terminal, if you type **git hist** you will see that the last commit was on the **feature-update-readme** file.

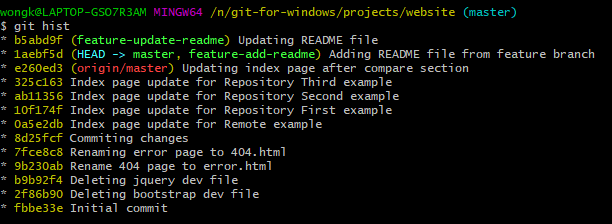
**NOTE:** The HEAD tag is placed next to the branch that we are currently on.



* Switch to the master branch by typing **git checkout master**.



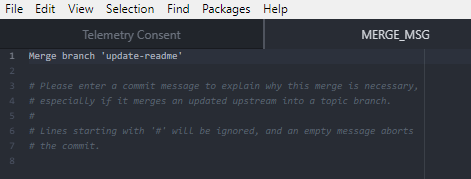
* If you type **git hist** again, you will notice that the last commit was for the **feature-update-readme** but that we are currently on the **master, feature-add-readme** commit as we have navigated to that branch.



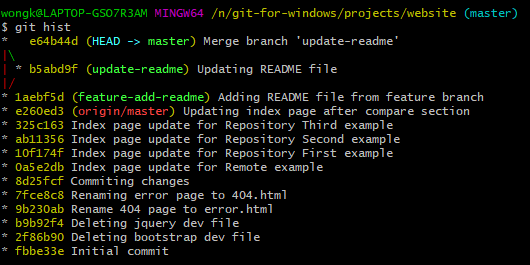
* We are now going to change the name of one of the branches.
* We will change the **feature-update-readme** branch to just **update-readme**.
* The syntax to use is **git branch -m <name of branch you want to change name> <new name of branch>**
* Type the following in the terminal to change the branch name.
* **git branch -m feature-update-readme update-readme**



* To perform the merge WITHOUT the Fast forward option, meaning that we will have atom open up with the MSG log file, type the following.
* **git merge –no-ff update-readme**



* Just save and close the file and the merge will proceed.
* If you type in **git hist** you should see the merging of the update-readme branch into the master branch.



**LECTURE 65 – Automatic Merges**

* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Asdfasfas
* Fas
* Fa
* Sdfa
* Sdf
* Asf
* Adsf
* Asdf
* Ads
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Asdfasfas
* Fas
* Fa
* Sdfa
* Sdf
* Asf
* Adsf
* Asdf
* Ads
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Asdfasfas
* Fas
* Fa
* Sdfa
* Sdf
* Asf
* Adsf
* Asdf
* Ads
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Asdfasfas
* Fas
* Fa
* Sdfa
* Sdf
* Asf
* Adsf
* Asdf
* Ads
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Asdfasfas
* Fas
* Fa
* Sdfa
* Sdf
* Asf
* Adsf
* Asdf
* Ads
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Asdfasfas
* Fas
* Fa
* Sdfa
* Sdf
* Asf
* Adsf
* Asdf
* Ads
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Asdfasfas
* Fas
* Fa
* Sdfa
* Sdf
* Asf
* Adsf
* Asdf
* Ads
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Asdfasfas
* Fas
* Fa
* Sdfa
* Sdf
* Asf
* Adsf
* Asdf
* Ads
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Asdfasfas
* Fas
* Fa
* Sdfa
* Sdf
* Asf
* Adsf
* Asdf
* Ads
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Asdfasfas
* Fas
* Fa
* Sdfa
* Sdf
* Asf
* Adsf
* Asdf
* Ads
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Asdfasfas
* Fas
* Fa
* Sdfa
* Sdf
* Asf
* Adsf
* Asdf
* Ads
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Asdfasfas
* Fas
* Fa
* Sdfa
* Sdf
* Asf
* Adsf
* Asdf
* Ads
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Asdfasfas
* Fas
* Fa
* Sdfa
* Sdf
* Asf
* Adsf
* Asdf
* Ads
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Asdfasfas
* Fas
* Fa
* Sdfa
* Sdf
* Asf
* Adsf
* Asdf
* Ads
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Asdfasfas
* Fas
* Fa
* Sdfa
* Sdf
* Asf
* Adsf
* Asdf
* Ads
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Asdfasfas
* Fas
* Fa
* Sdfa
* Sdf
* Asf
* Adsf
* Asdf
* Ads
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf
* Asdfasfas
* Fas
* Fa
* Sdfa
* Sdf
* Asf
* Adsf
* Asdf
* Ads
* Fasd
* Fasd
* Fas
* Fas
* Dfa
* Dsf
* Adsf
* adsf