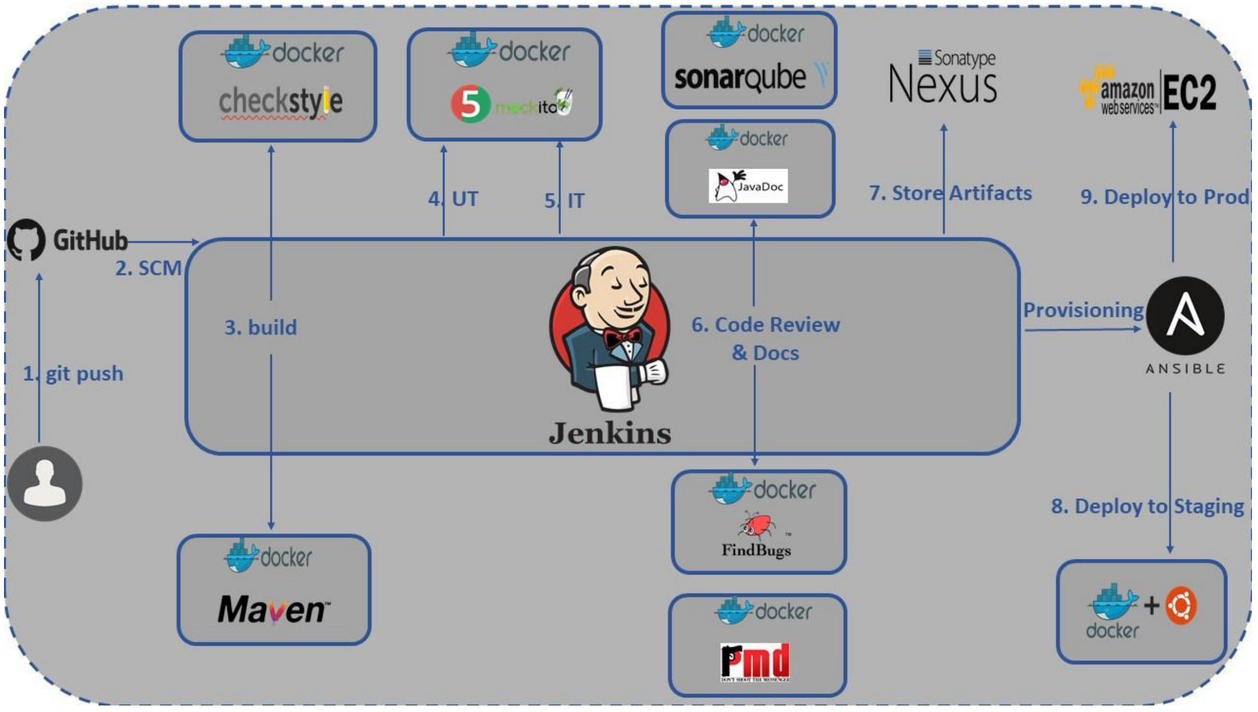
DevOps



DEV and OPS

This is the service that connects the Developers and the Operations

Every company out there is now a software company.

The companies have their own developers who are tasked with developing the codes.

When a software is developed, it undergoes the DEV testing and QA, where the quality assurance is done for the software.

Software can be developed using the following methodologies:

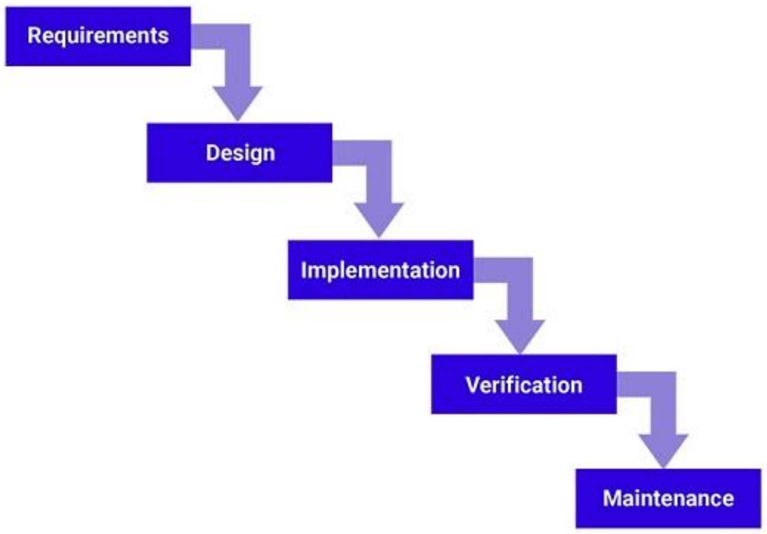
Waterfall methodology

Waterfall methodology – was the first to exist. You meet with the developer and agree on what you want. Developer starts working on the software and after 2 to 3 months he delivers the completed software. Any error will be fixed after the software has been delivered. By the time the software is delivered, in most cases, some of its functionalities will always be outdated especially in the fast-paced technological world of today.

The waterfall method is a rigid linear model that consists of sequential phases (requirements, design, implementation, verification, maintenance) focusing on distinct goals.

Each phase must be 100% complete before the next phase can start.

There’s usually no process for going back to modify the project or direction until delivery is done and client test the software.



Benefits

The linear nature of the waterfall development method makes it easy to understand and manage.

Projects with clear objectives and stable requirements can best use the waterfall method.

Disadvantages

The waterfall development method is **often slow and costly** due to its rigid structure and tight controls.

Error prone because the software is only tested after it has been delivered to the client.

Agile methodology

In this methodology, the software is developed in little chunks. You start by developing the parts that are beginning features which in most cases are the most important features. For instance, if you are developing a software, you can start by creating the login or the registration form.

In this methodology the delivery can be done in sprints of 2 to 3 weeks. In this methodology, most clients are closer to the developers because the delivery can be done frequently to allow the communication between developers and the clients.

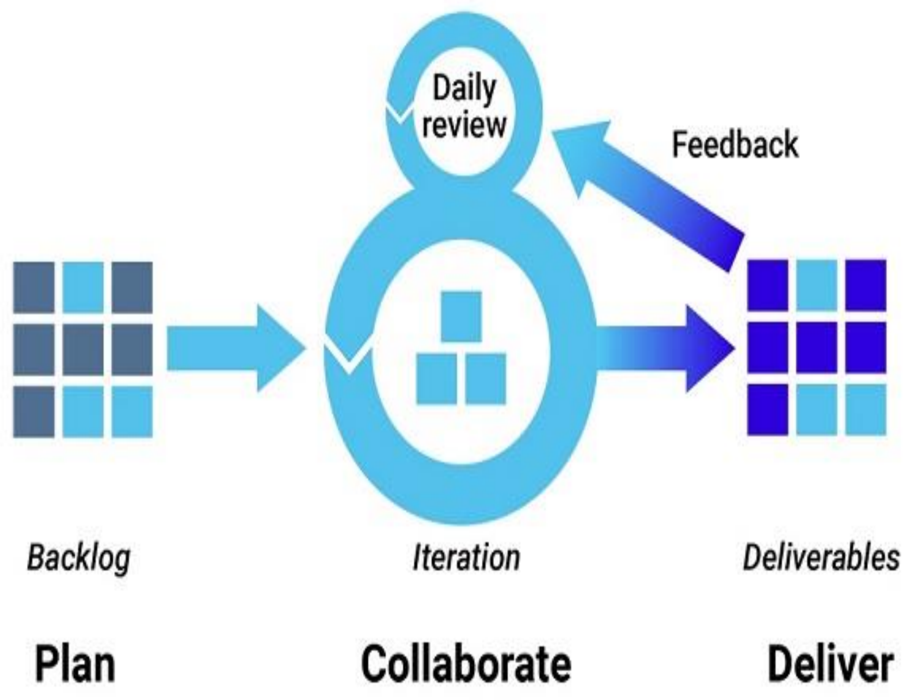
Most companies are in the agile methodology.

You know it is agile when you hear words like we deploy the code in 2 weeks frequencies.

Teams use the agile development methodology to minimize risk (such as bugs, cost overruns, and changing requirements) when adding new functionality.

In all agile methods, teams develop the software in iterations that contain mini-increments of the new functionality.

There are many different forms of the agile development method, including scrum, crystal, extreme programming (XP), and feature-driven development (FDD).



Benefits: The primary benefit of agile software development is that it allows software to be released in iterations.

Iterative releases improve efficiency by allowing teams to find and fix defects and align expectation early on.

They also allow users to realize software benefits earlier, with frequent incremental improvements.

Disadvantages: Agile development methods rely on real-time communication, so new users often lack the documentation they need to get up to speed.

require a huge time commitment from users and are labor intensive because developers must fully complete each feature within each iteration for user approval.

Feedback is a must to continue to the next stage of the development.

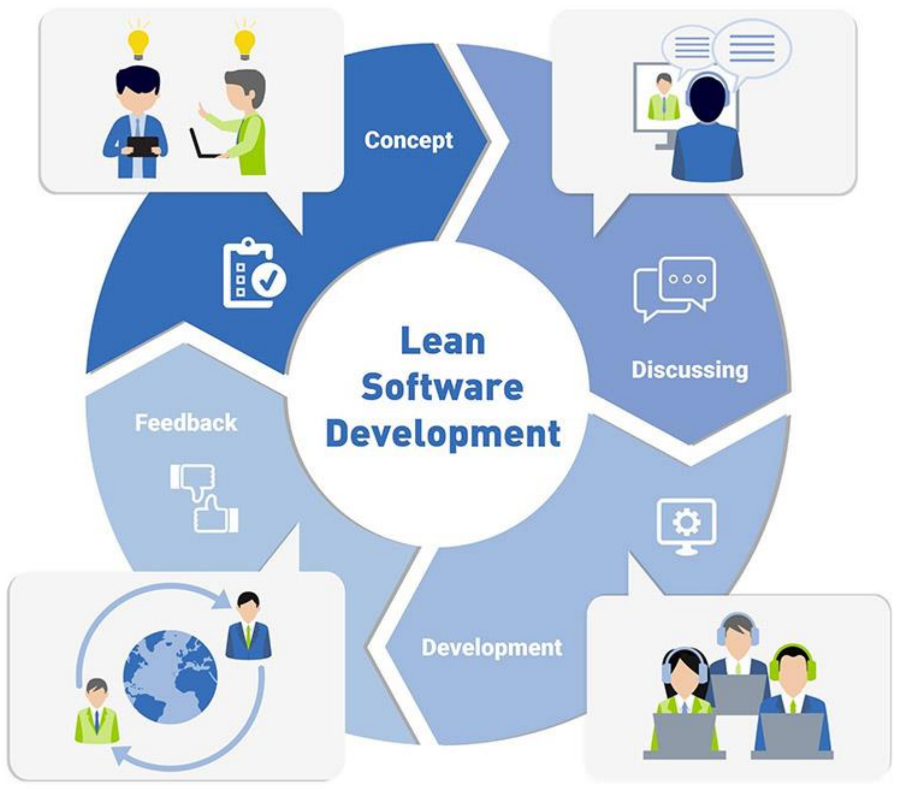
Lean Methodology

What is Lean Software Development (LSD)?

Lean Software Development (LSD) is an agile framework based on **optimizing development time and resources, eliminating waste, and ultimately delivering only what the product needs.**

The aim of the lean methodology is to maximize the resources and reduce any waste in the form of resources and time.

The Lean approach is also often referred to as the Minimum Viable Product (MVP) strategy, in **which a team releases a bare-minimum version of its product to the market,** learns from users what they like, don't like and want to be added, and then iterates based on this feedback.



We come up with a concept, we discuss the concept, we develop the concept and then we collect the feedback and the process repeat itself until we have a complete project.

DevOps deployment methodology

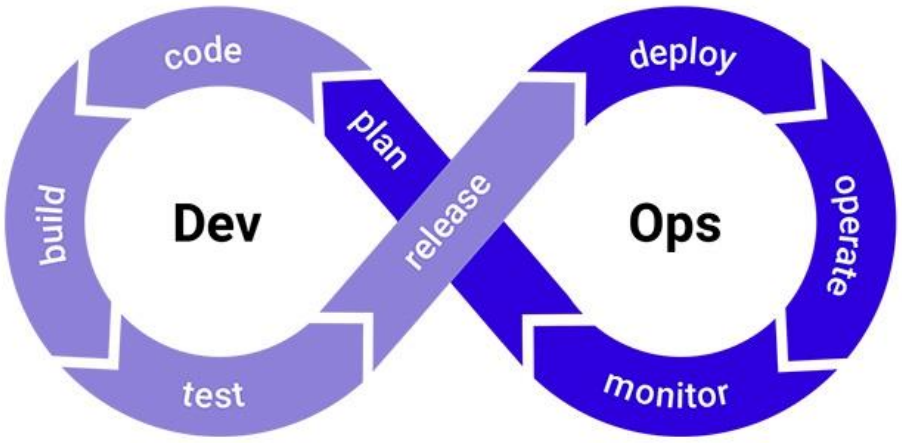
DevOps methodology is important because it moves the developers and the operations closer to each other.

DevOps is not just a development methodology but also a set of practices that supports an organizational culture.

DevOps deployment centers on organizational change that enhances collaboration between the departments responsible for different segments of the development life cycle, such as development, quality assurance, and operations.

DevOps is more than just a software development methodology.

It allows the collaboration of the different departments in the companies such as development, this is the department for developers, quality assurance (QA) where the testing for the quality standards and assurance is carried out to ensure the software meets the required criteria by the company and the operation department that oversees the testing and the deployment of the software.



Benefits:

DevOps is focused **on improving time to market, lowering the failure rate of new releases, shortening the lead time between fixes, and minimizing disruption while maximizing reliability.**

To achieve this, DevOps organizations aim to **automate continuous deployment** to ensure everything happens smoothly and reliably.

Companies that use DevOps methods benefit by significantly reducing time to market and improving **customer satisfaction, product quality, and employee productivity and efficiency.**

Disadvantages: Even considering its benefits, there are a few drawbacks to DevOps:

- Some customers do not want continuous updates to their systems.
- Some industries have regulations that require extensive testing before a project can move to the operations phase

Why docker is important in DevOps

What is difference between containerization and virtualization?

Deploying different images or applications to run on the same OS system which is the docker engine.

Virtualization is the running of different OS system applications on the same OS system server.

Docker for DevOps

Docker is a platform that turns out to be a perfect fit for the DevOps ecosystem.

- It is developed for software companies that are struggling to pace up with the changing technology, business, and customer requirements.
- The benefits Docker offers to the DevOps environment has made it an irreplaceable tool in the toolchain.
- The reason for Docker being very good for DevOps is its benefits and use cases of containerizing the applications that support the development and quick release.
- DevOps is primarily used to overcome **'Dev' and 'Ops' problems,** and Docker seems to solve most of them, with the main one being **that it can work on any machine**.
- Thus, it allows all the teams to collaborate and work effectively and efficiently regardless of the environment they are deploying the codes.
- Docker allows you to make inevitable development, production, and staging environments, thereby providing you seamless control overall changes.

- If you want to return to the previous version, you can do that anytime, as all the environments become more alike.
- Docker guarantees that if a feature is functioning in the **development environment**, it will work in the **production and staging environment as well.**

DevOps Three Distinctive Environments

Active development happening in the **development environment**

Testing and QA (quality assurance) happening in the **staging environment**

Production where the Code is being pushed to production or being released in **the production environment** once QA is complete in the staging environment.

DevOps is based on two key concepts:

Cross-Functional Teams: There is no separation between **development, operations, database, and testing teams. It's a single and coordinated IT department.**

Automation: Container-based technologies aligned with the microservice architecture have changed how software is built, allowing full automation of all the software development cycles, including development, testing, deployment, and monitoring.

At the core of DevOps are **The Three Ways**. The Three ways are the core guidelines and agile development best practices key for a business transformation.

The principle of Flow, which accelerates the delivery of work from Development to Operations, to our customers.

The principle of Feedback: which enables teams to create safer systems of work.

The principle of Continual Learning and Experimentation, which fosters a high-trust culture and a scientific approach to organizational improvement as part of daily work.

In many organizations, the engineering and operations teams work apart, which can generate misalignments, a less efficient development process, a lot of back and forth, and communication problems.

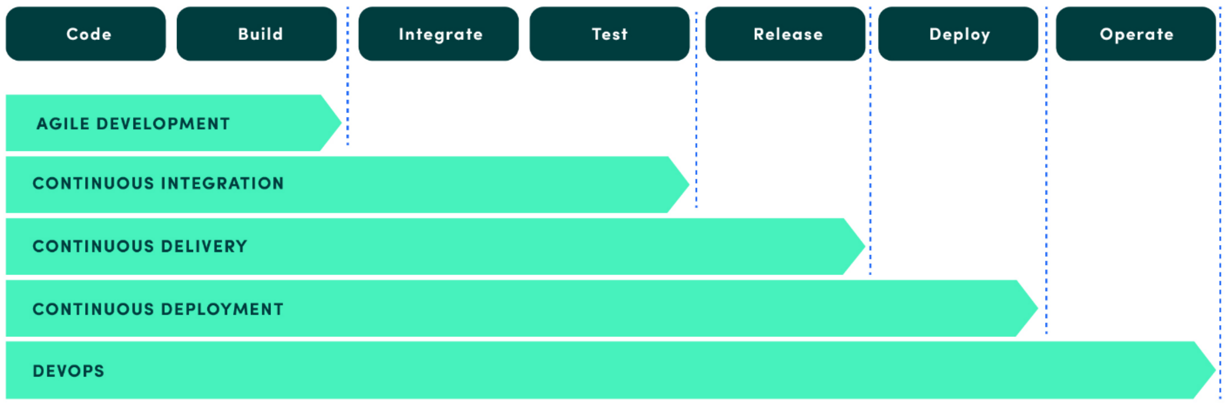
Adopting a DevOps mindset helps companies **achieve shorter development cycles and a better alignment with business objectives.**

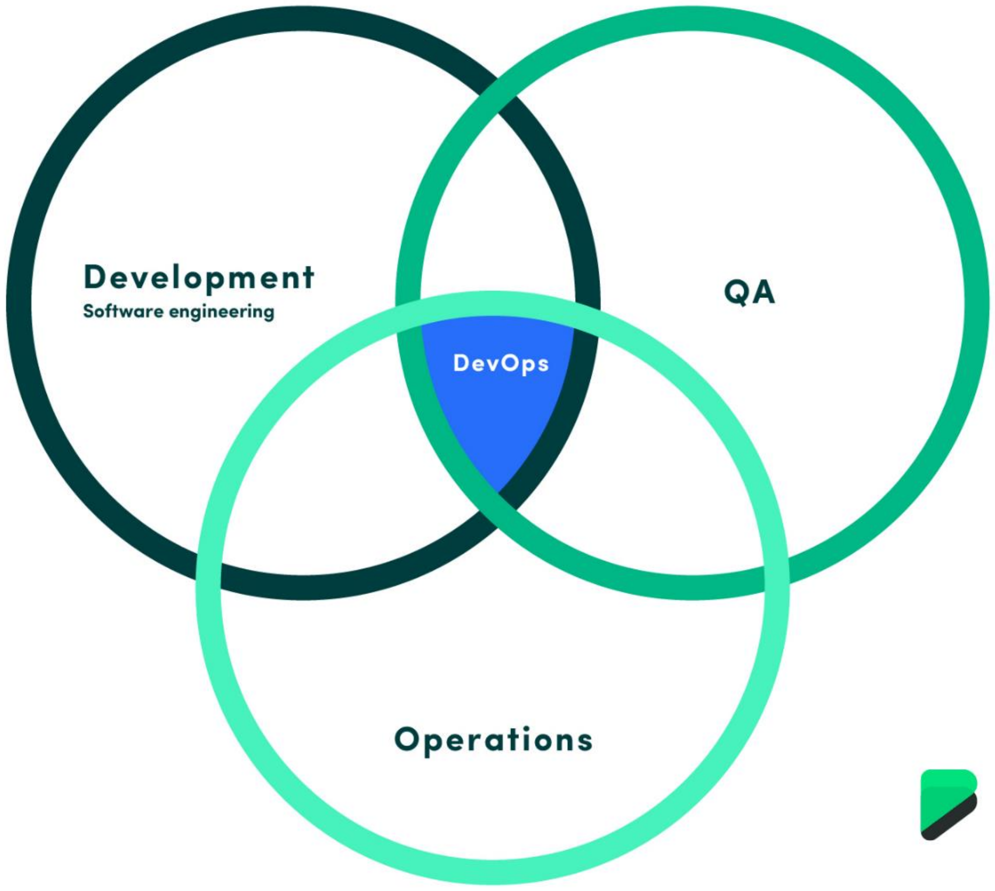
DevOps also enables teams to improve collaboration with stakeholders and to implement automation throughout the delivery process.

As a software development practice, DevOps has some distinctive objectives:

- Improve deployment frequency
- Allow the software product to achieve a faster time to market
- Decrease the failure rate of new releases
- Shorten the lead time between fixes

- Enable infrastructure to move as quickly as developers need it to





CONTINUOUS INTEGRATION

Continuous integration (CI) is the practice of **frequently integrating (combining) new code with the existing code repository**.

Continuous Integration should occur frequently enough **that no errors can arise without developers noticing them** and correcting them immediately.

When it comes to releasing the code, **CI helps to save a lot of time.**

Continuous integration is often the first down the road toward DevOps maturity. This practice includes 10 essential tasks:

- Maintain a code repository – Github
- **Automate the build**
- Make the build self-testing
- Everyone commits to the baseline every day
- Every commit (to baseline) should be built

- Keep the build fast
- Test in a clone of the production environment
- Make it easy to get the latest deliverables
- Everyone can see the results of the latest build
- Automate deployment

CONTINUOUS DELIVERY

Continuous delivery (CD) is a software engineering approach that allows teams to produce software in short cycles, ensuring that the software can be reliably released at any time.

CD is defined as the act of creating, testing, and releasing code frequently in small increments.

Continuous Delivery includes a series of practices designed to ensure that code can be rapidly and safely deployed to production by delivering new iterations to a production-like environment and ensuring applications and services function as expected through rigorous automated testing.

Since every change is delivered to a staging environment, you can **have confidence the application can be deployed to production with a push of a button when the business is ready.**

The benefits of continuous delivery are:

- Reduced Deployment Risk: since you are deploying smaller changes, there's less to go wrong, and it's easier to fix possible issues.
- Believable Progress: Continuous Delivery allows teams to track progress by tracking iterations deployed into a production (or production-like) environment.
- User Feedback: the biggest risk to any software effort is to build something that isn't useful. The earlier and more frequently you get working software in front of real users, the quicker you get feedback to find out how valuable it really is.

CONTINUOUS DEPLOYMENT

Continuous Deployment is the advanced evolution of Continuous Delivery. **This practice allows teams to deploy software iterations into production without any human intervention.**

Every change that passes the automated tests is deployed automatically, resulting in many production deployments every day.

A team that works with Continuous Development never deploys any code without testing.

New code is usually deployed to a small number of users, and there's an automated feedback loop that monitors the quality and usages before the code gets to the next level.

Continuous Deployment should be the goal of most companies that are not constrained by regulatory or other requirements.

Build Automation

Build automation is critical for DevOps.

Binary Code Vs. Source Code

Binary releases contain computer readable version of the application, meaning it is compiled. Source releases contain human readable version of the application, meaning it must be compiled before it can be used. The source release is the raw, uncompiled code. You could read it yourself.

Build automation is the process of **automating the retrieval of source code,** compiling it into **binary code, executing automated tests, and publishing it into a shared, centralized repository. Build automation is critical to successful DevOps processes.**

Source code management (SCM) is used to track modifications to a source code repository. SCM tracks a running history of changes to a code base and helps resolve conflicts when merging updates from multiple contributors. SCM is also synonymous with Version control.

Git is a distributed version control system that records different versions of a file (or set of files). However, GitHub is mainly a hosting platform for hosting Git repositories online.

Build automation enables Continuous Integration (CI). So, the role of CI in the automated build process is that CI uses build automation to verify check-ins and enable teams to detect issues early.

Because of its relationship with CI, build automation also makes **Continuous Testing (CT) and Continuous Delivery (CD) — possible**

5 Benefits of Build Automation

There are five main benefits of build automation.

Increases Productivity

Build automation ensures fast feedback. This means your developers increase productivity. They will spend less time dealing with tools and processes and more time delivering value.

Accelerates Delivery

Build automation helps you accelerate delivery. That is because it eliminates redundant tasks and ensures you find issues faster, so you can release faster.

Improves Quality

Build automation helps your team move faster. That means you will be able to find issues faster and resolve them to improve the overall quality of your product and avoid bad builds.

Maintains a Complete History

Build automation maintains a complete history of files and changes. That means you will be able to track issues back to their source.

Saves Time and Money

Build automation saves time and money. That is because build automation sets you up for CI/CD, increases productivity, accelerates delivery, and improves quality.

How to Automate the Build Process

Here's how to automate the build process:

- Write the code.
- Commit code to a shared, centralized repository — such as Perforce Helix Core.
- Scan the code using tools such as static analysis.
- Start a code review.
- Compile code and files.
- Run automated testing.
- Notify contributors to resolve issues.

Repository: A directory or storage space where your projects can live. Sometimes GitHub users shorten this to “repo.”

It can be local to a folder on your computer, or it can be a storage space on GitHub or another online host.

You can keep code files, text files, image files, you name it, inside a repository.

Automated Build Tools

Automated build tools will help you ensure build automation.

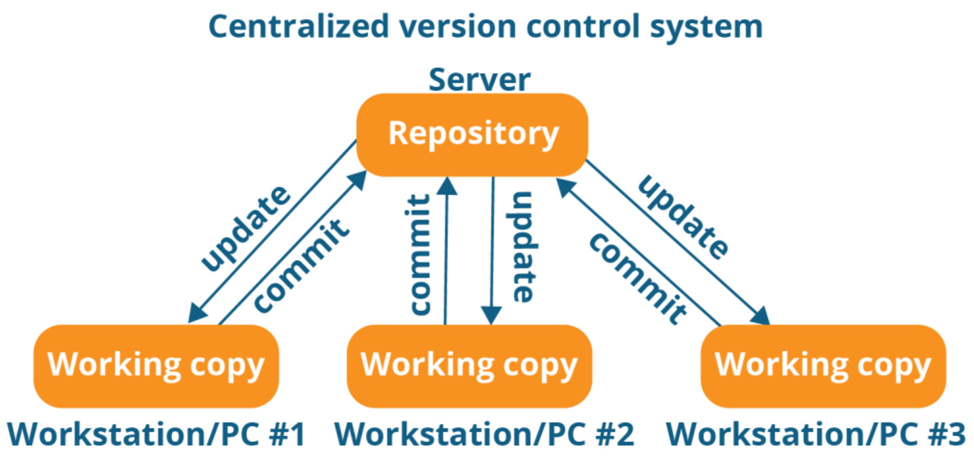
Build runners are critical for automation. **These tools help you automate the process of building, testing, and deploying code.**

Example: Jenkins

Jenkins is a popular build runner. Many teams automate their CI/CD pipeline with Jenkins.

But there’s one more DevOps pipeline tool that you’ll need for build automation. That’s version control.

Version control provides the shared, centralized repository that’s essential for build automation.



What is version control?

Version control can be defined as a management system that manages the changes made in the project until the end.

What a version control does is to create and save a snapshot of the entire program whenever changes are made in the code.

These snapshots are called as different versions. Changes can be of any type such as adding a new file to our project, editing some other file, or replacing the source code.

One of the main purposes of the version control system is that we can roll back to any previous version if we are not satisfied with our current version

Why version control?

The version control system provides a shared workspace, so, if any developer has made changes in the code of the same project, everyone will get notified about changes that are made by the one.

By using version control, we always have a safe backup. If the central server crashes, a backup is always available on all the local servers.

The version control system provides you with all the information needed to make a proper analysis of the entire project.

It informs about what changes are made, when those changes are made, and how much time did it take.

What are Git and Github?

Git is the most used version control system tool. It is a great medium to collaborate with other developers in order to make a decent and appropriate project.

Github is a web-based repository. It provides you a great place to use git.

Why are Git and GitHub used?

It is the best way to build a project if you are working with a team.

Team members can work on files and easily include their changes in the master branch of the project.

This allows multiple people to work on the same files at the same time.

As it is a version control system, every team member has a copy of the entire project on their local repository.

Every version of the project is available, so if you want to revert some changes, it can be done easily using git.

It is a very good backup option for your project.

If any team member or every team member accidentally loses the data on their local workspace, there is no need to worry as it is saved safely on the cloud.

GIT

https://git-scm.com/

GIT is a version control system that is used for tracking changes made to the files.

GIT is very beneficial because of the following:

1. It is decentralized or distributed version control which means that many developers can work on same project **without necessarily being in the same network**. Can also be used by developers in the same network.
2. Coordinates work or projects between multiple developers and track all the changes made, who made the changes and when they are made and how long it took to make the changes.
3. Whenever you make any changes, you can always revert to the previous version at any time provided you were committed to the repository.
4. Whenever you work with GIT you are going to have a local repository in the local machine, you work on and make changes and then you can push the local repository to the remote such as GITHUB or bitbucket.
5. You will have both local and remote repos
6. Locally you don't need an internet to work on the local repository in your local machine but to push to the remote repository, such as bitbucket, you must have internet.

Concepts of GIT

- Keep tracks of all the changes in the code history by taking **snapshots** of your files.
- Snapshots of the code files can be taken by making a "commit" with simple command.
- You can always visit any snapshot at any time.
- You can stage files before committing by simply using the add command. The files in the GIT are very secure and safe.

Why is it important to "Commit" local repository to remote repository?

It is very vital to make a Commit of the remote repository so that other developers can access the repository and pull the repository to their local machines and start working and making the necessary changes on it.

Important Commands

1. $ git init - // initialize the local Git repository and create a **.git folder** in the project you are working on. The folder is hidden by default, and I believe is for security purpose.

After initializing the folder as a local repository, you can now start to run other commands.

No need to go to this folder.

2. $ git add <file> . This command is important and will add the specified files to the staging area of the index as they wait for more commands.
3. $git status – this will check the status of the working tree or what you have in the staging area. This command displays the difference between the working tree and the staging area of the index.
4. $git commit – this commit changes in the index and takes everything one has in the staging area of the index to the local repository when one is ready.

Commands for the remote repository like github or bitbucket

5. $git push will push the local repository to the remote repository. You will have to add credentials and all that.
6. $git pull – pull the latest from remote repository.
7. $git clone – will clone repository from Github into a new directory and download to your machine.

Installing GIT

Ubuntu

# Apt install git

Centos 7

# Yum install git

Mac

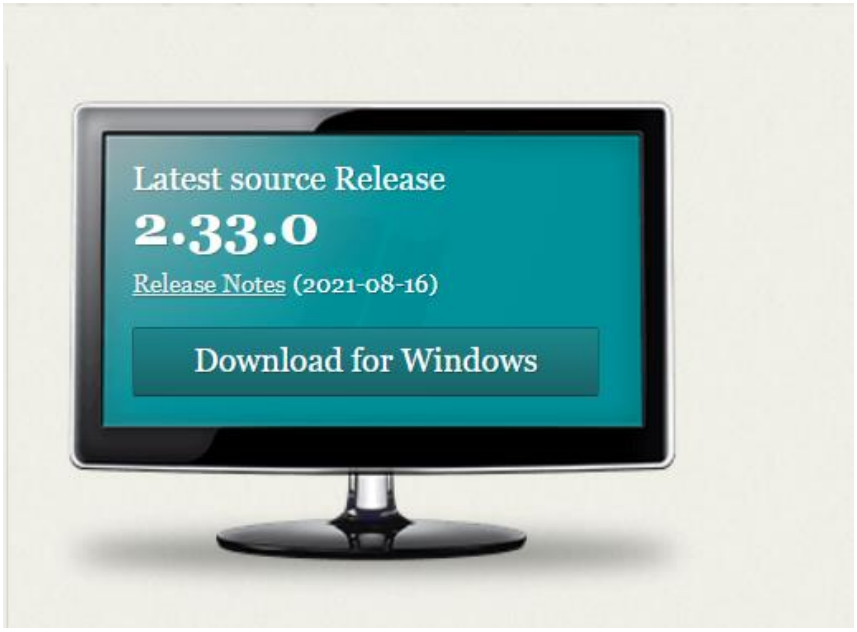
http://git.scm.com/download/mac

Windows

http://git.scm.com/download/win

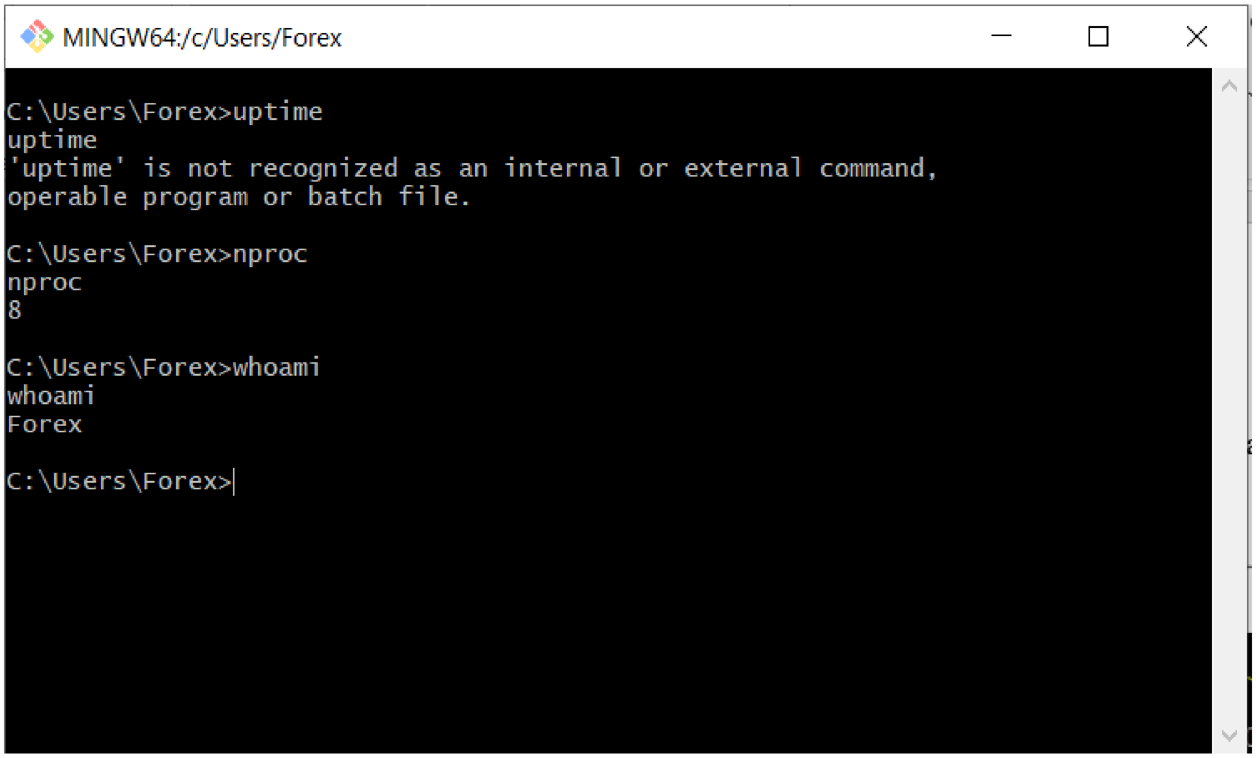
Let getting Started by Installing GIT

https://git-scm.com/



Click download for windows

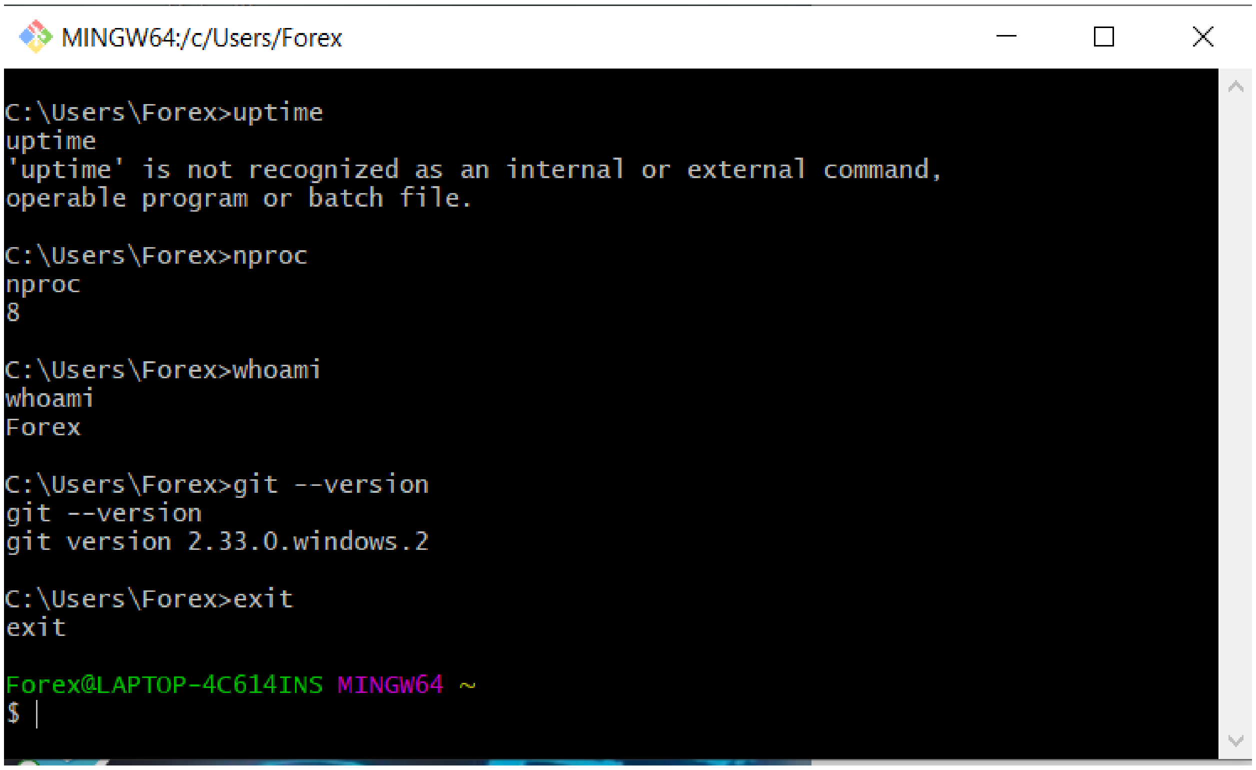
If you type $ cmd, window's terminal will open



MINGW64:/c/Users/Forex

```
C:\Users\Forex>uptime
uptime
'uptime' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\Forex>nproc
nproc
8

C:\Users\Forex>whoami
whoami
Forex

C:\Users\Forex>
```

Type EXIT to return from CMD to GIT terminal



MINGW64:/c/Users/Forex

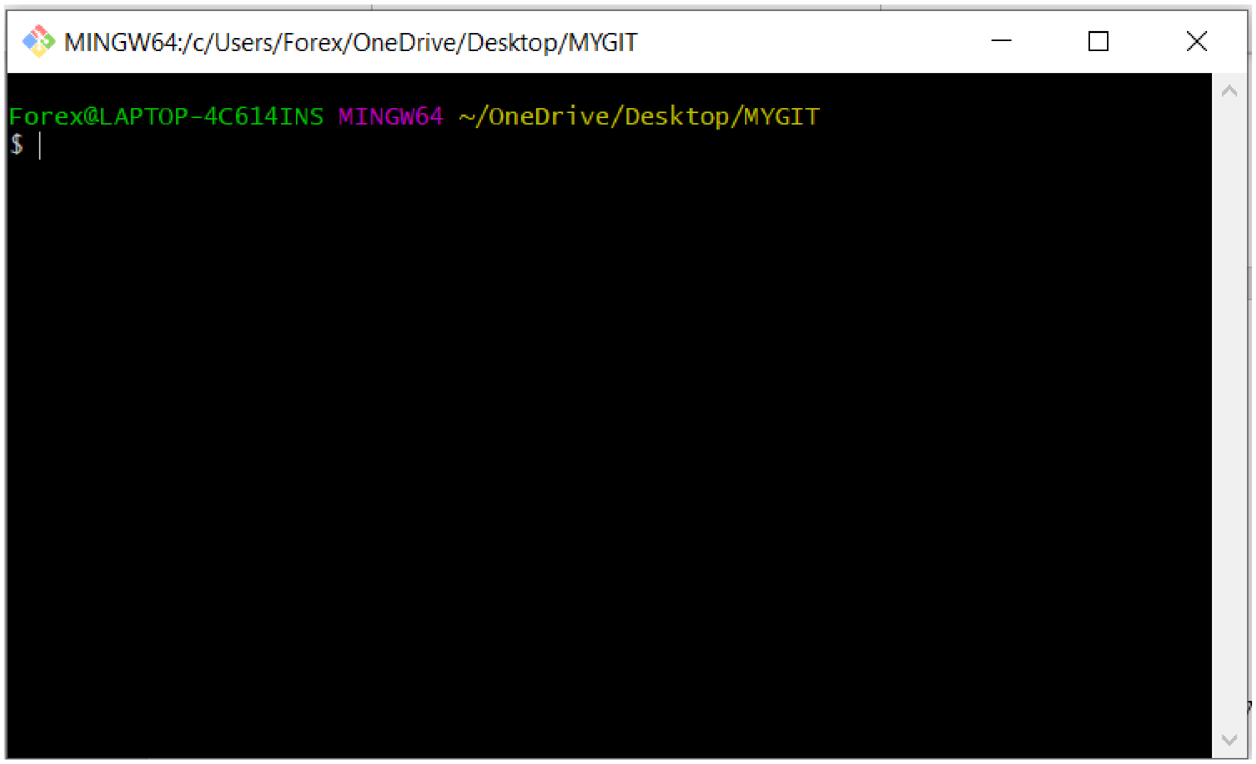
```
C:\Users\Forex>uptime
uptime
'uptime' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\Forex>nproc
nproc
8

C:\Users\Forex>whoami
whoami
Forex

C:\Users\Forex>git --version
git --version
git version 2.33.0.windows.2

C:\Users\Forex>exit
exit

Forex@LAPTOP-4C614INS MINGW64 ~
$ 
```

Let us create a project

Open a new folder called MYGIT

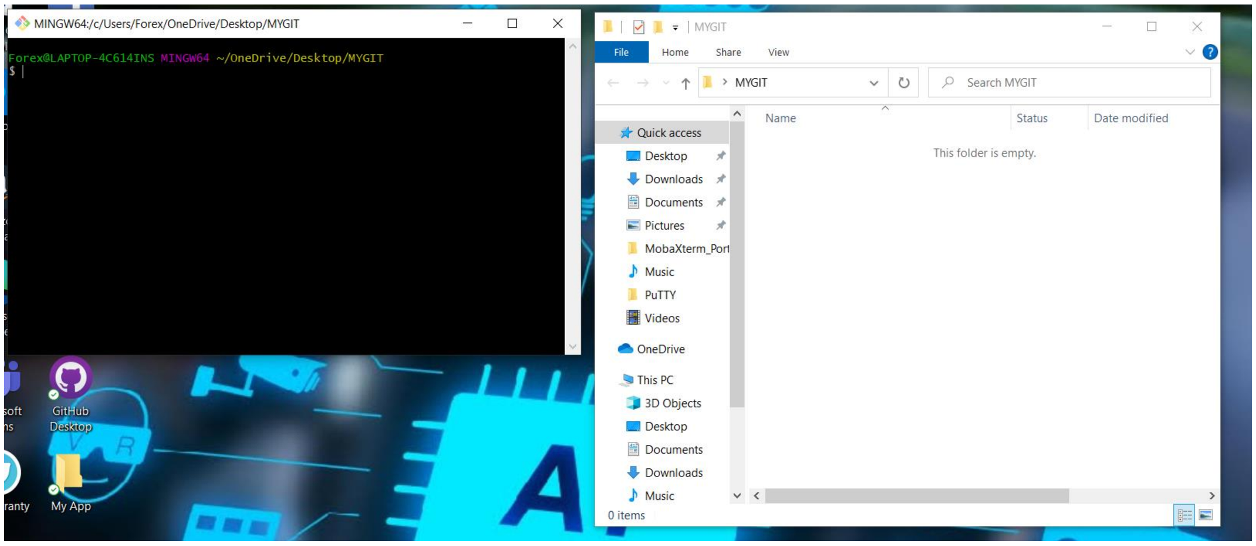
Right click on the folder created and saved in the desktop

Since you have GIT, you can now open the folder with GIT SCM/VCM


MINGW64:/c/Users/Forex/OneDrive/Desktop/MYGIT

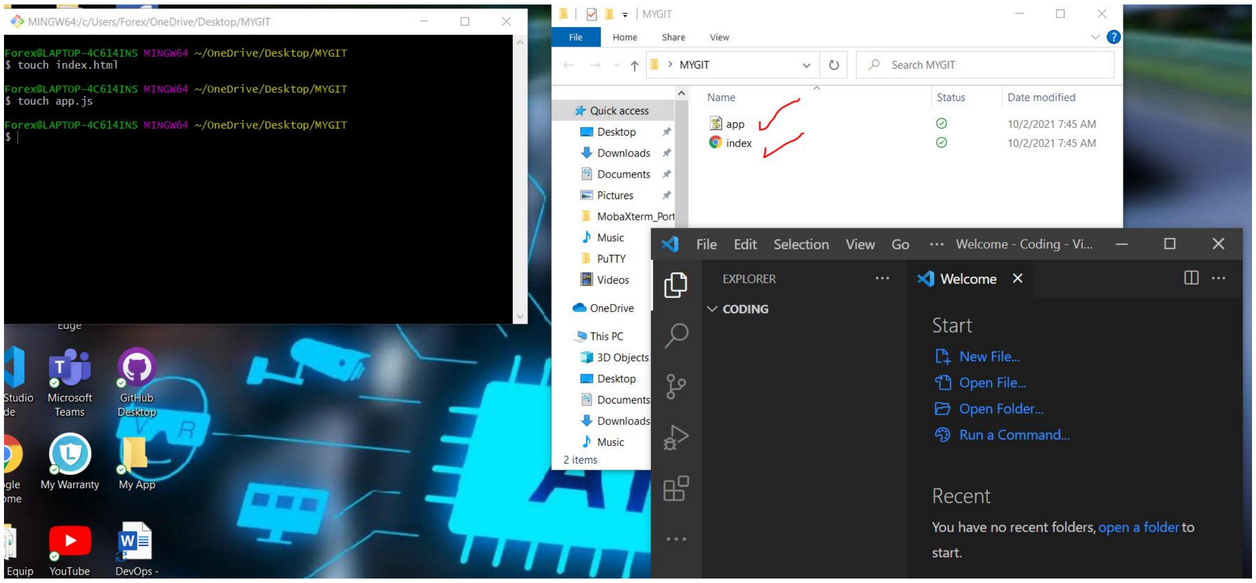
```
Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT
$ |
```

Open both the folder and GIT Bash terminal connected to the folder you created


MINGW64:/c/Users/Forex/OneDrive/Desktop/MYGIT

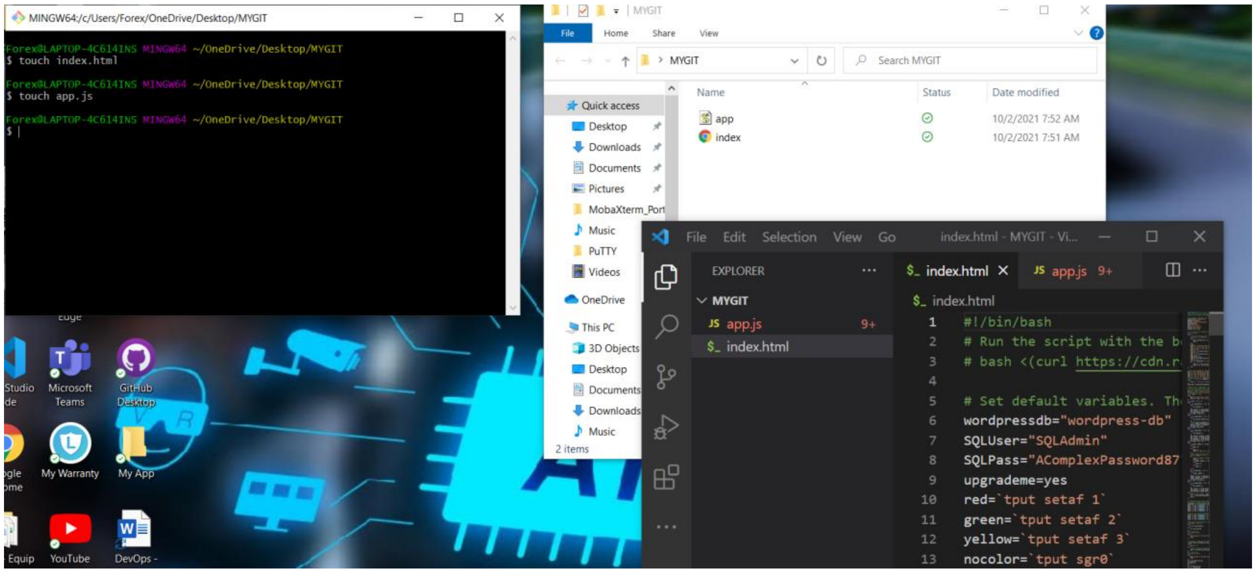
```
Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT
$ |
```

You will see the file being created as you run the touch command



```
Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT
$ touch index.html

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT
$ touch app.js

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT
$ |
```

Open your visual studio Code and add the folder as a project



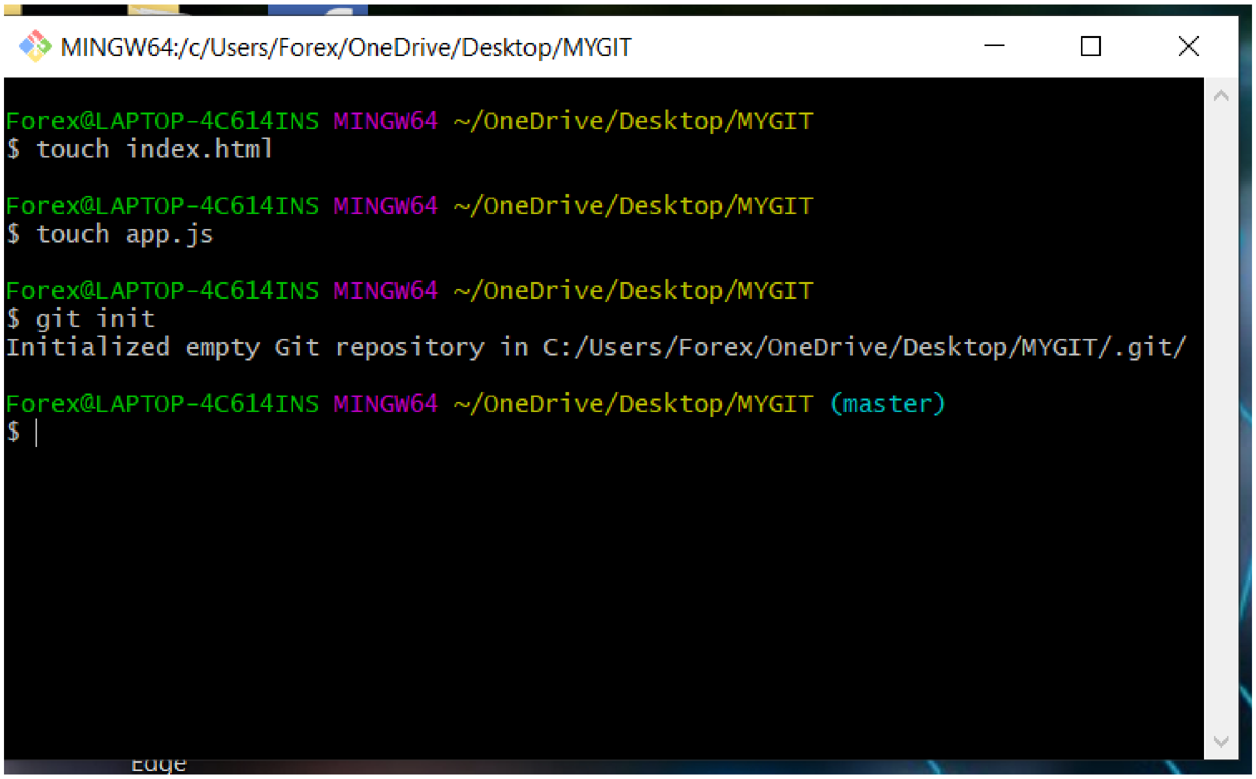
```
Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT
$ touch index.html

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT
$ touch app.js

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT
$ |
```

```
#!/bin/bash
# Run the script with the b
# bash <(curl https://cdn.r

# Set default variables. Th
wordpressdb="wordpress-db"
SQLUser="SQLAdmin"
SQLPass="AComplexPassword87
upgrademe=yes
red=`tput setaf 1`
green=`tput setaf 2`
yellow=`tput setaf 3`
nocolor=`tput sgr0`
```

After putting some contents in the file, we can now run the GIT commands

# Initialize the folder as a new git repository and a .git folder in my local directory.

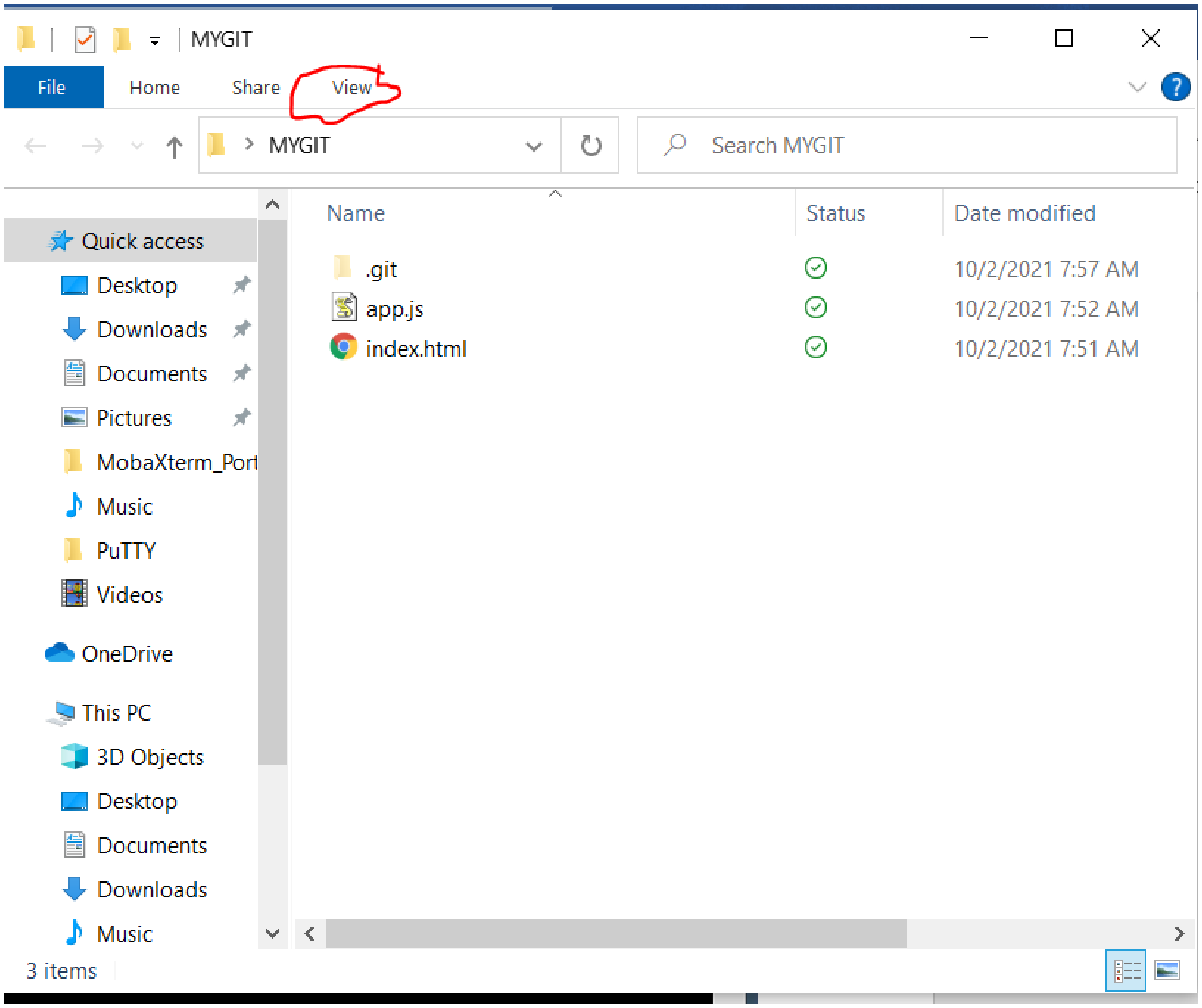
You can enable the hidden file capability on the windows to see the new folder that has been initialized to become local repository on your local machine



```
Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT
$ touch index.html

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT
$ touch app.js

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT
$ git init
Initialized empty Git repository in C:/Users/Forex/OneDrive/Desktop/MYGIT/.git/

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ |
```

How to view the .git folder created in your local machine

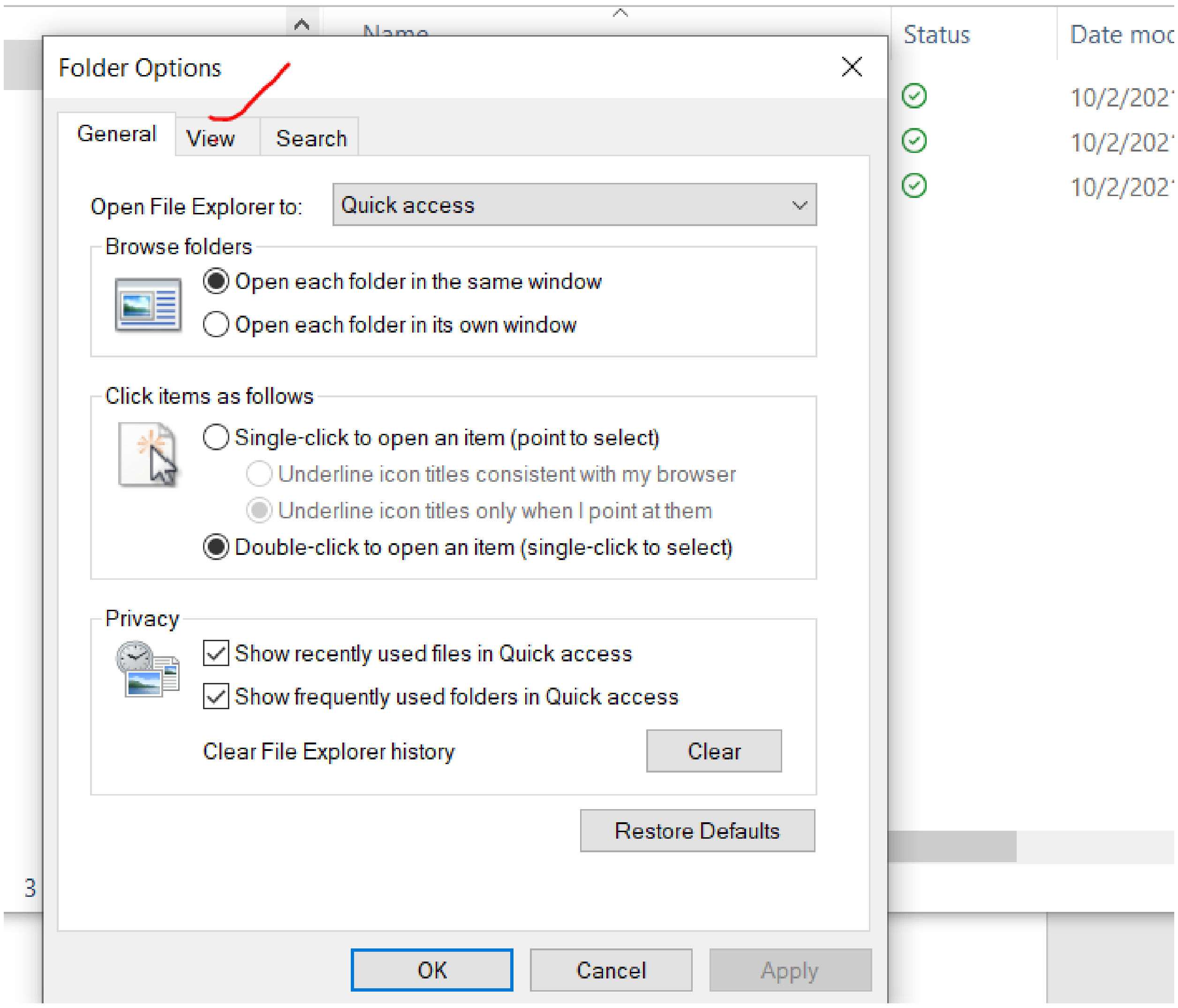
Open your folder explorer



Name	Status	Date modified
.git	✓	10/2/2021 7:57 AM
app.js	✓	10/2/2021 7:52 AM
index.html	✓	10/2/2021 7:51 AM

Then go to Options

Click view again



Folder Options

General | View | Search

Open File Explorer to: Quick access

Browse folders

[✓ Open each folder in the same window]
Open each folder in its own window

Click items as follows

Single-click to open an item (point to select)
Underline icon titles consistent with my browser
[✓ Underline icon titles only when I point at them]
[✓ Double-click to open an item (single-click to select)]

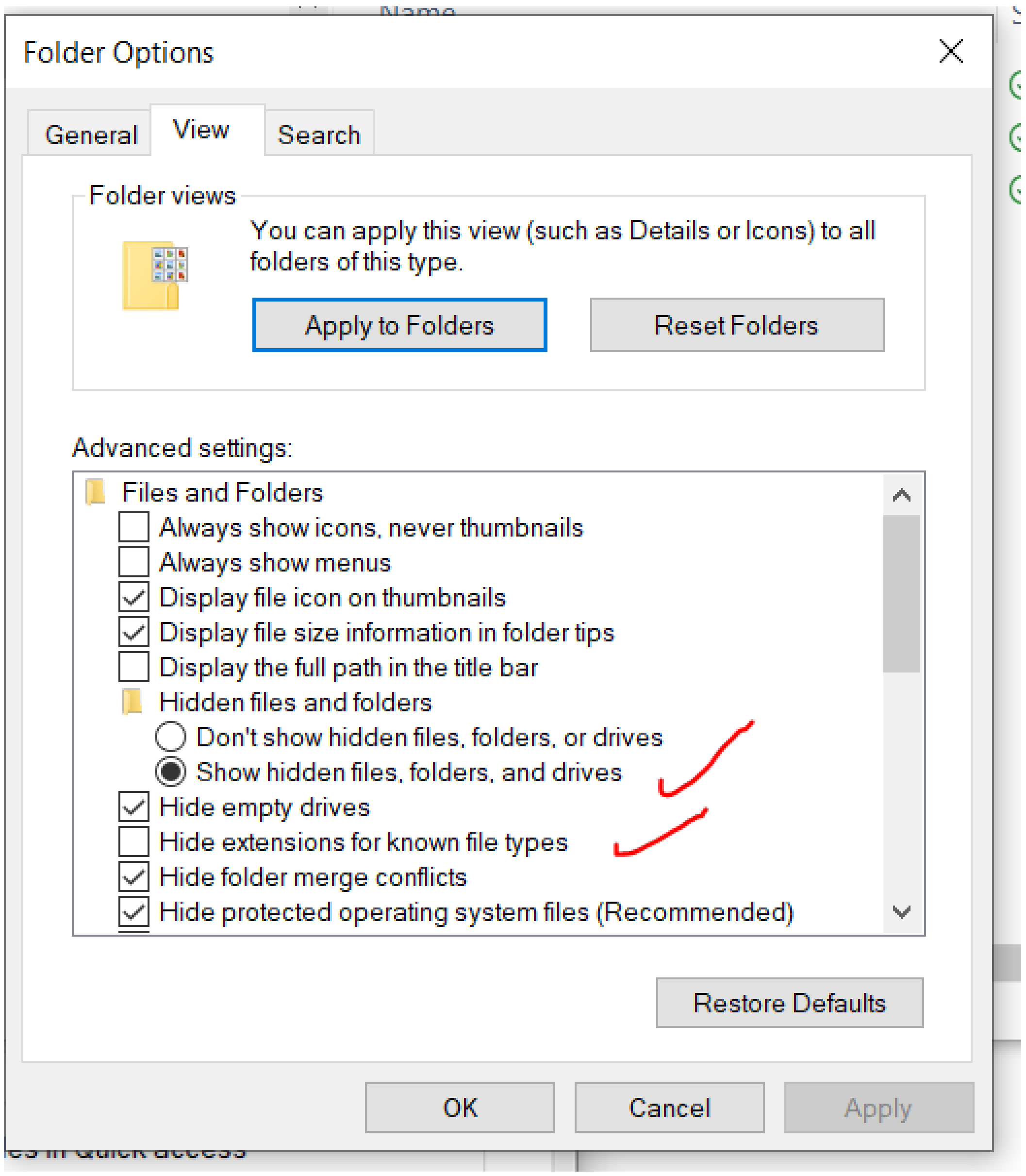
Privacy

☑ Show recently used files in Quick access
☑ Show frequently used folders in Quick access

Clear File Explorer history — Clear

Restore Defaults

OK | Cancel | Apply



Folder Options

General | View | Search

Folder views

You can apply this view (such as Details or Icons) to all folders of this type.

Apply to Folders | Reset Folders

Advanced settings:

- Files and Folders
  - ☐ Always show icons, never thumbnails
  - ☐ Always show menus
  - ☑ Display file icon on thumbnails
  - ☑ Display file size information in folder tips
  - ☐ Display the full path in the title bar
  - Hidden files and folders
    - ○ Don't show hidden files, folders, or drives
    - ● Show hidden files, folders, and drives
  - ☑ Hide empty drives
  - ☐ Hide extensions for known file types
  - ☑ Hide folder merge conflicts
  - ☑ Hide protected operating system files (Recommended)

Restore Defaults

OK | Cancel | Apply

You should be able to see your .git local repository in your machine

After initializing the local commands, we can now start to use the GIT commands

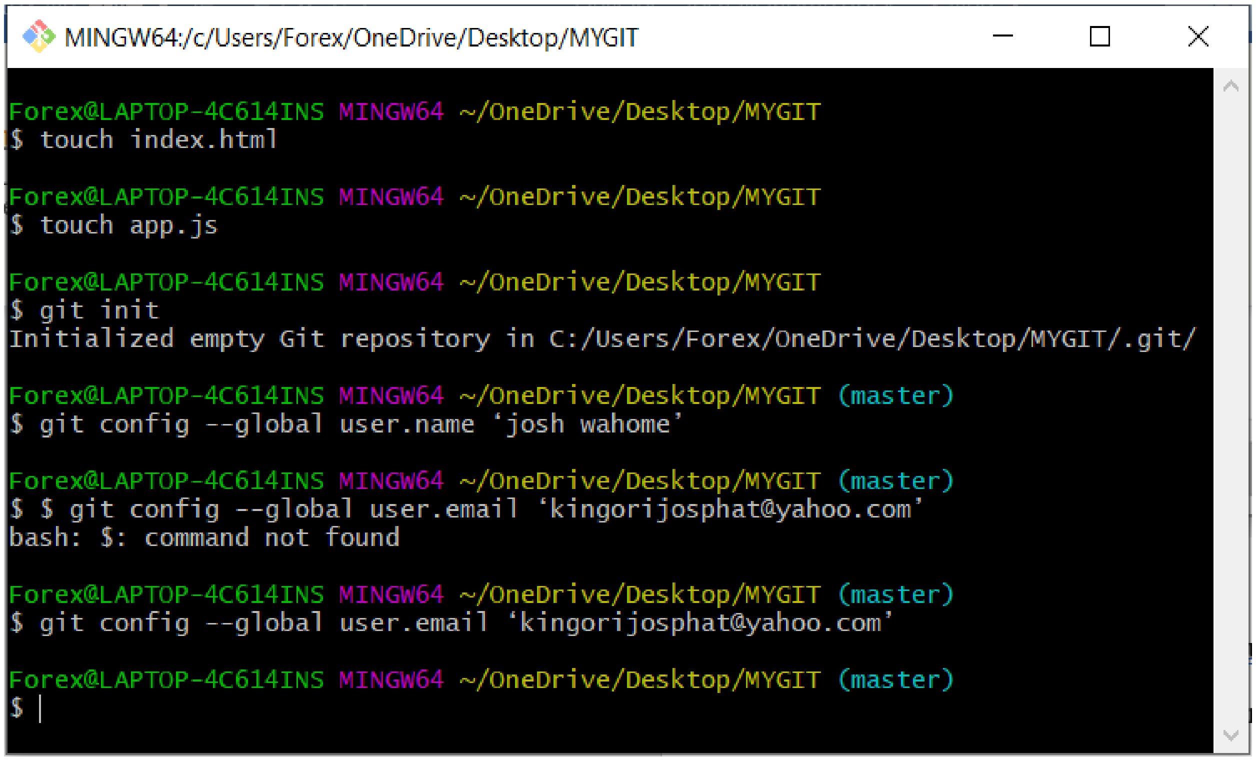
Before you can start anything, it is important to add your username and email to git

Run the config command

```
$ git config  --global user.name ‘josh wahome’
```

Do the same for email

```
$ git config --global user.email ‘kingorijosphat@yahoo.com’
```



```
Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT
$ touch index.html

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT
$ touch app.js

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT
$ git init
Initialized empty Git repository in C:/Users/Forex/OneDrive/Desktop/MYGIT/.git/

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git config --global user.name 'josh wahome'

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ $ git config --global user.email 'kingorijosphat@yahoo.com'
bash: $: command not found

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git config --global user.email 'kingorijosphat@yahoo.com'

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ |
```

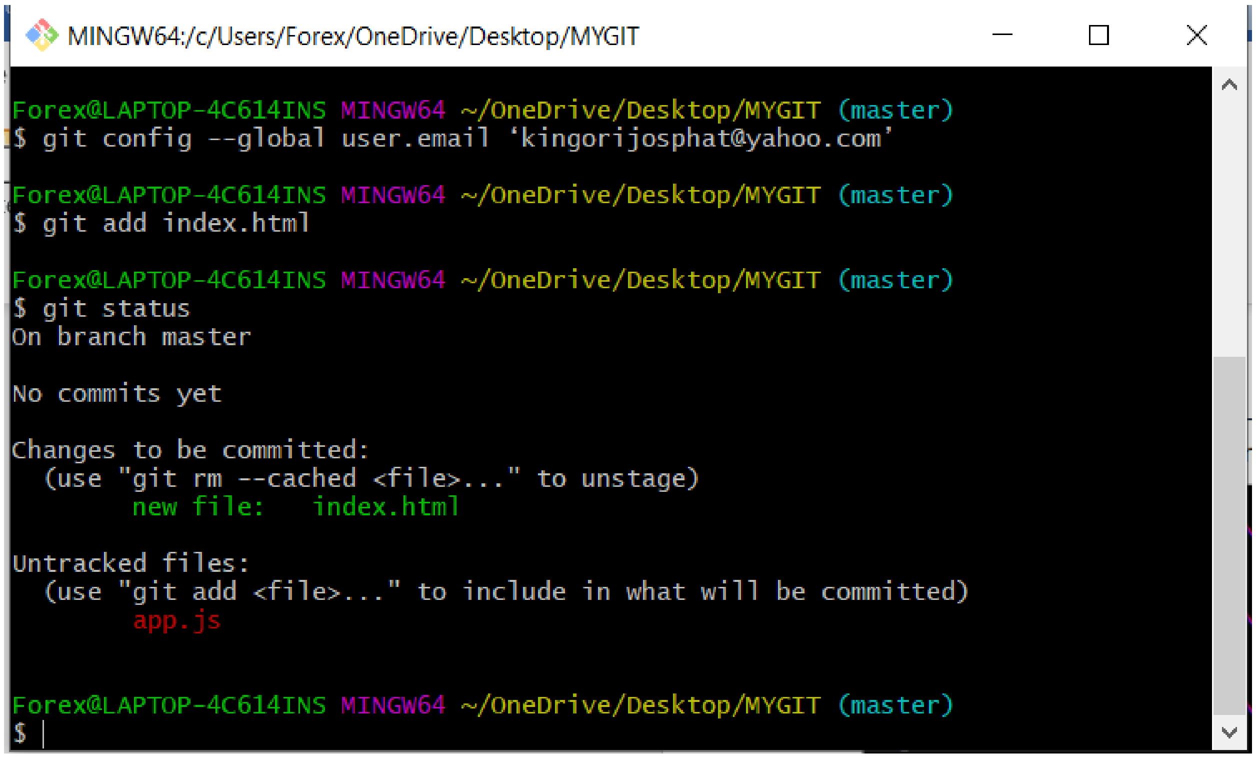
Now, let us add the html file to our staging area.

$ git add index.html

Let check the status of the added file in the staging area

$ git status

This is a command we are going to use a lot



MINGW64:/c/Users/Forex/OneDrive/Desktop/MYGIT

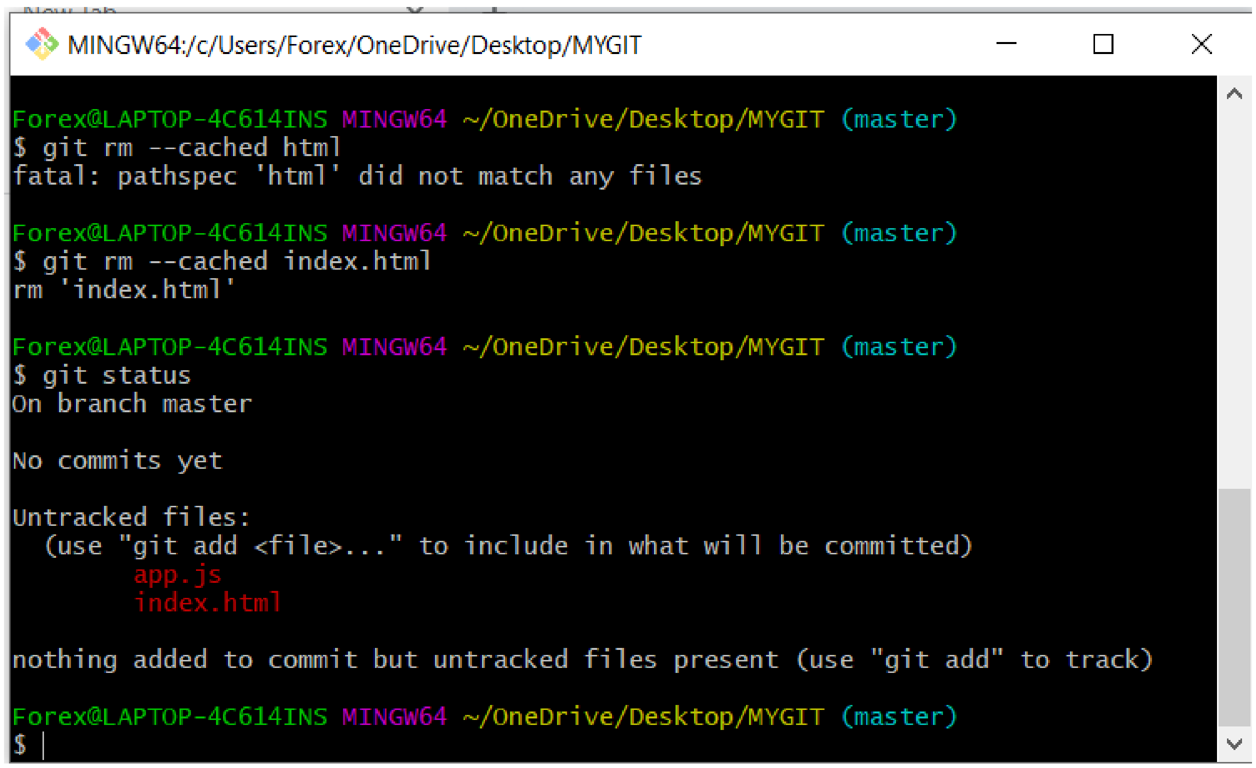
```
Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git config --global user.email 'kingorijosphat@yahoo.com'

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git add index.html

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git status
On branch master

No commits yet

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

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


Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ |
```

This is saying that the index.html file has changes to be committed.

Also, it is telling us that the app.js file has not been tracked and we should include it to be committed.

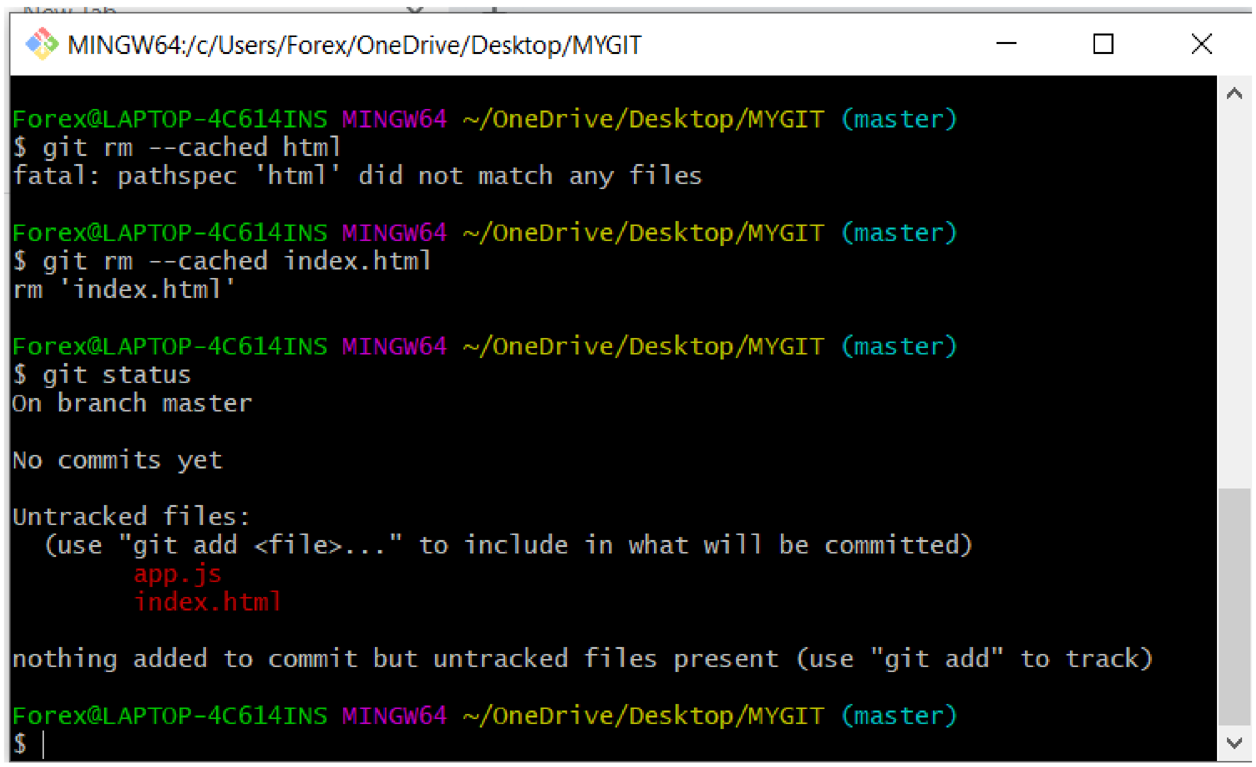
You can remove the committed file by running the command

$ git rm –cached index.html


MINGW64:/c/Users/Forex/OneDrive/Desktop/MYGIT

```
Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git rm --cached html
fatal: pathspec 'html' did not match any files

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git rm --cached index.html
rm 'index.html'

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git status
On branch master

No commits yet

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

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

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ |
```

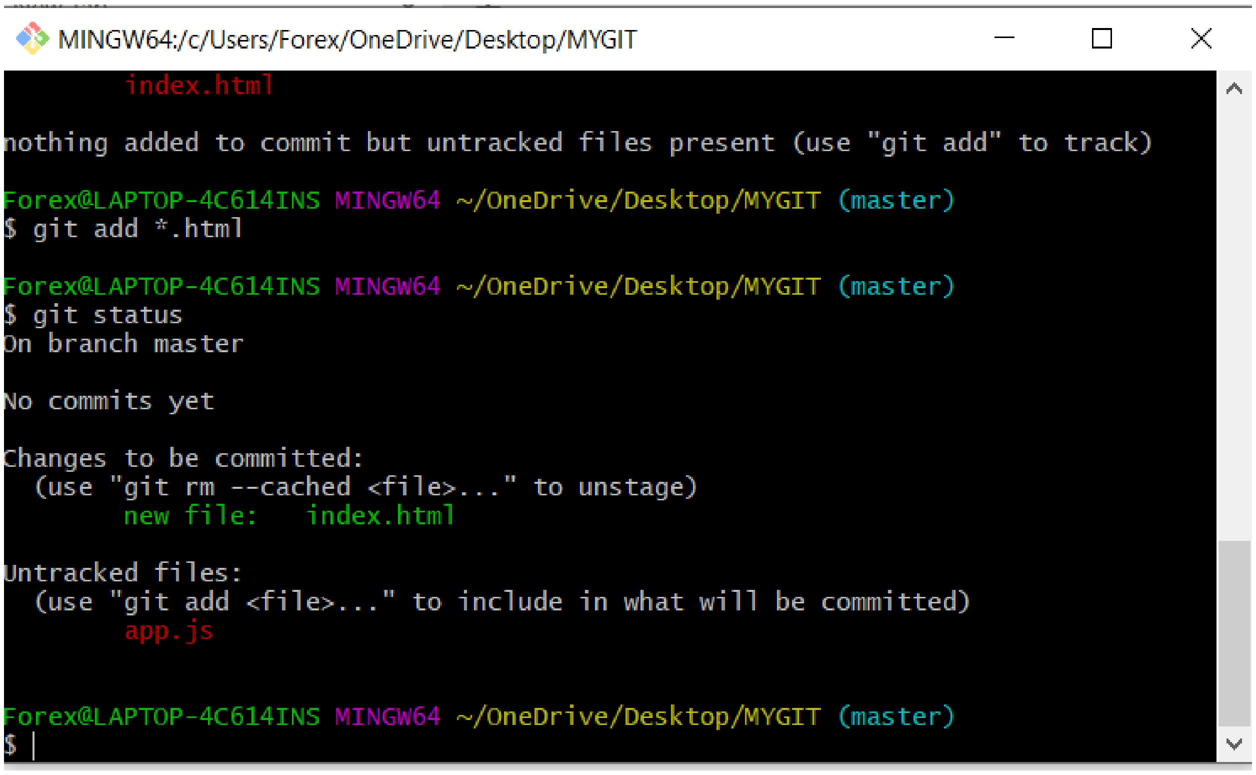
Then run $ git status


MINGW64:/c/Users/Forex/OneDrive/Desktop/MYGIT

```
Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git rm --cached html
fatal: pathspec 'html' did not match any files

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git rm --cached index.html
rm 'index.html'

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git status
On branch master

No commits yet

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

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

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ |
```

You can also add file by using the command

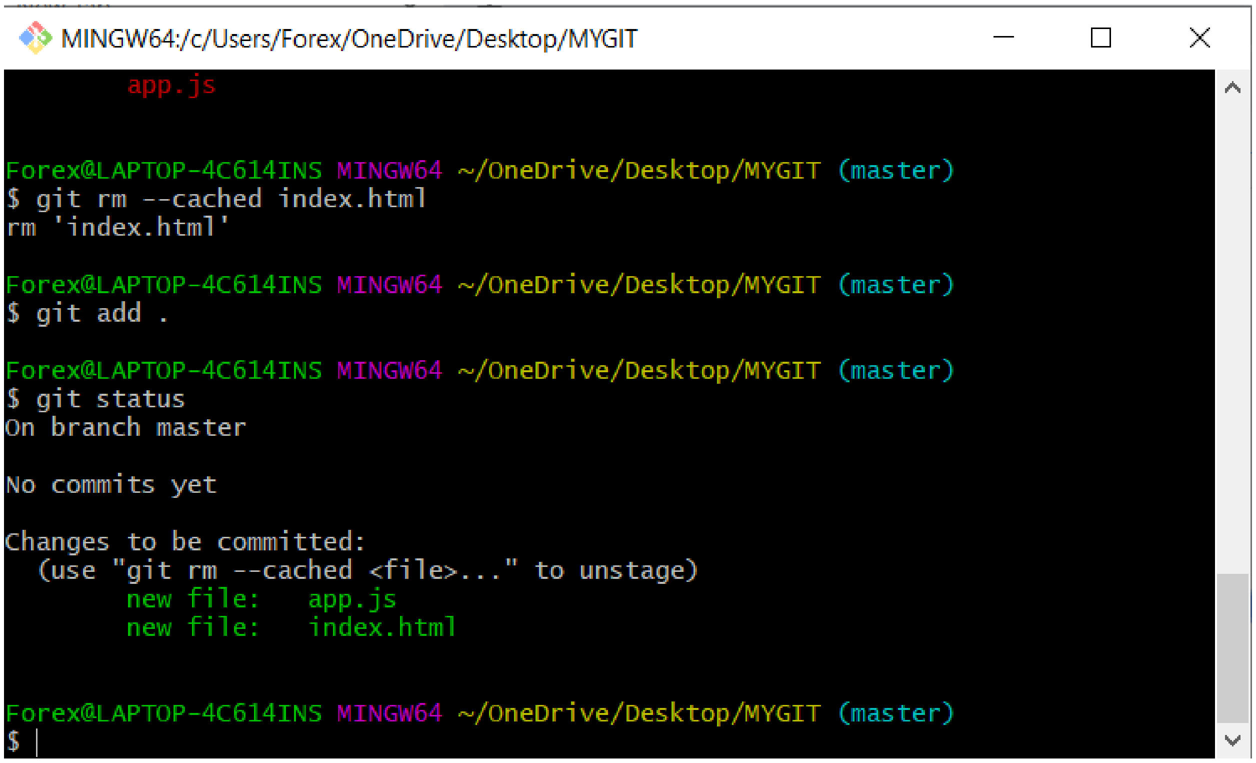
$git add *.html – this is going to add any html file we have



```
MINGW64:/c/Users/Forex/OneDrive/Desktop/MYGIT
        index.html

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

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git add *.html

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git status
On branch master

No commits yet

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

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


Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ |
```

You can add all the files you have in the .git repository by running the command

$ git add .

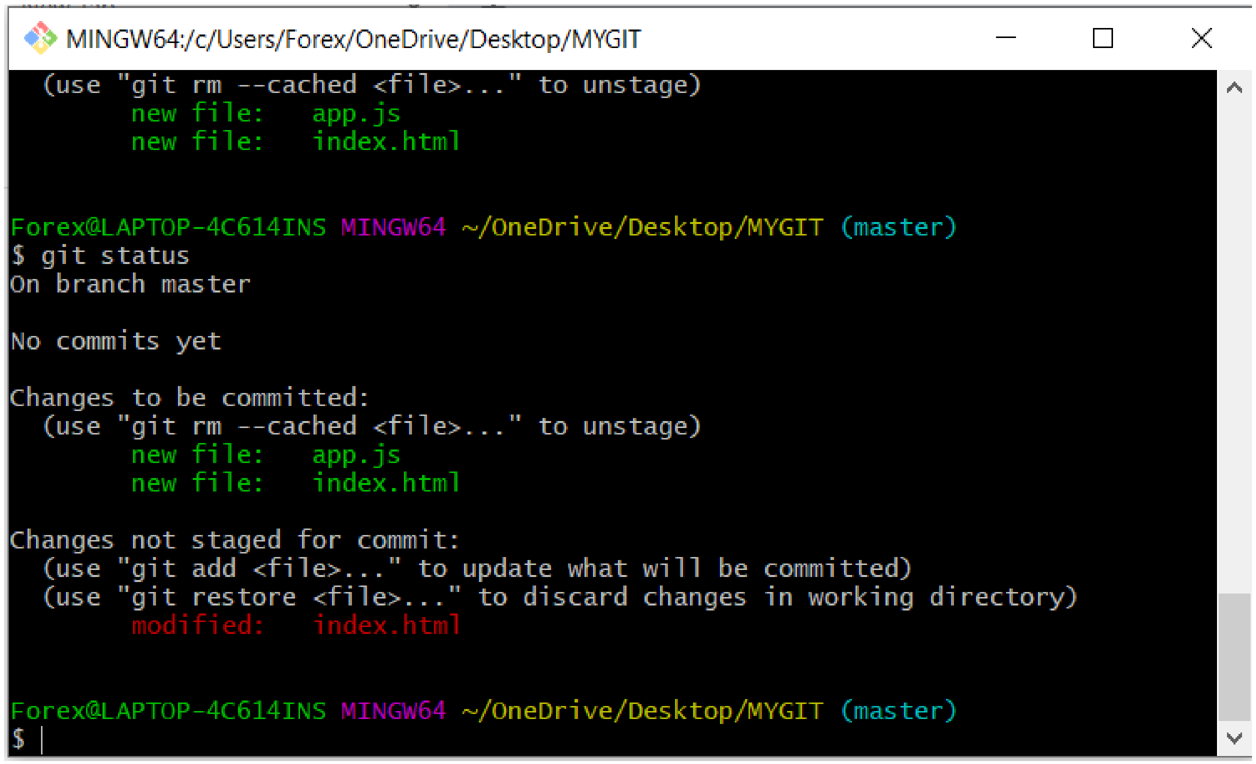


```
MINGW64:/c/Users/Forex/OneDrive/Desktop/MYGIT
        app.js


Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git rm --cached index.html
rm 'index.html'

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git add .

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   app.js
        new file:   index.html


Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ |
```

All files are in the staging area.

Let go to the index.html and make a simple change

Then, run the command $git status



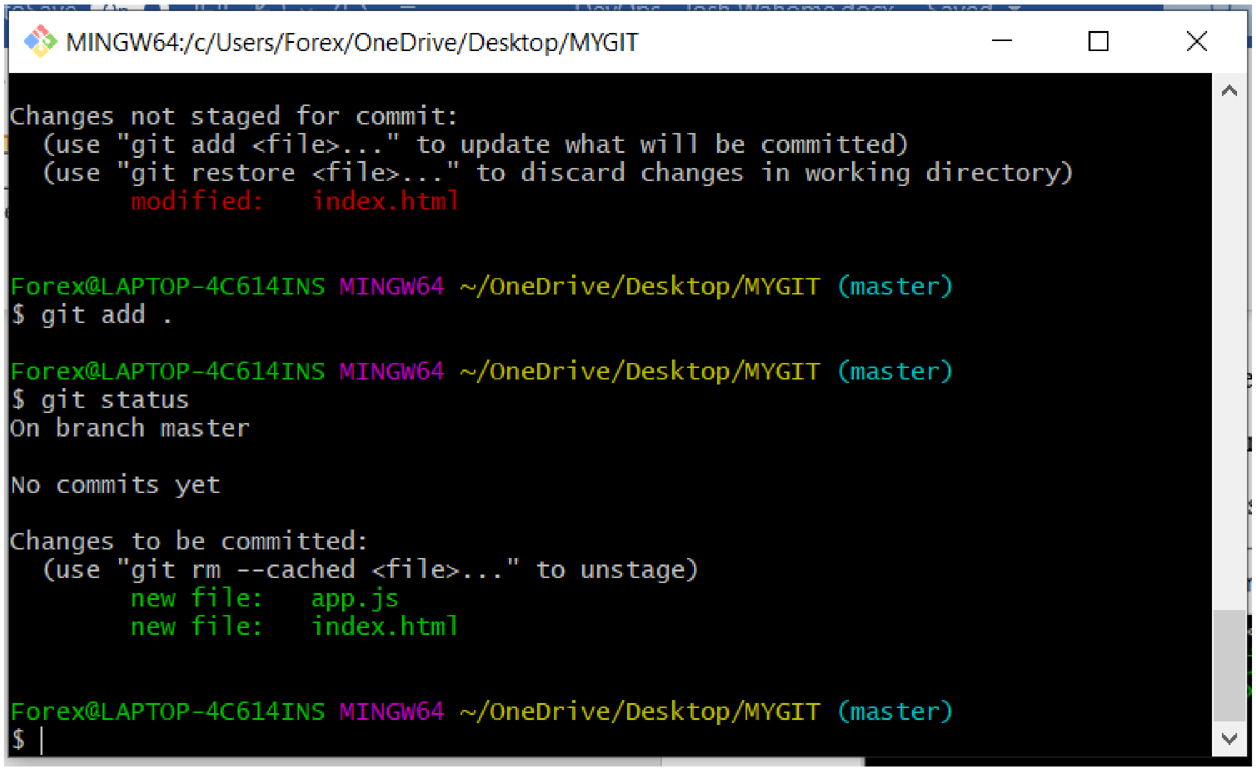
```
  (use "git rm --cached <file>..." to unstage)
        new file:   app.js
        new file:   index.html


Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   app.js
        new file:   index.html

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html


Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ |
```

It means we added some changes to the file while in staging area

To add the changes

Run $ git add .

Run $ git status and you will see the changes have been added.

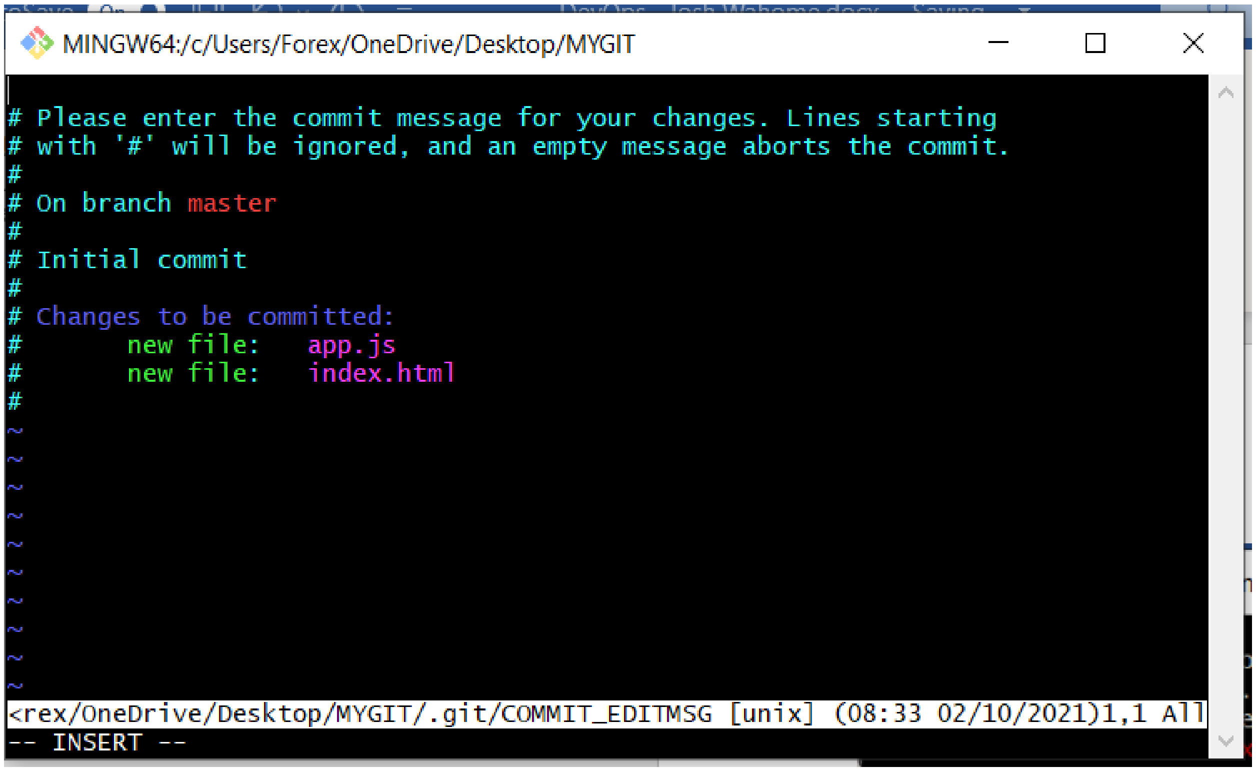

MINGW64:/c/Users/Forex/OneDrive/Desktop/MYGIT

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html


Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git add .

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   app.js
        new file:   index.html


Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ |
```

It is now back in the staging area

Let commit the files in the staging area

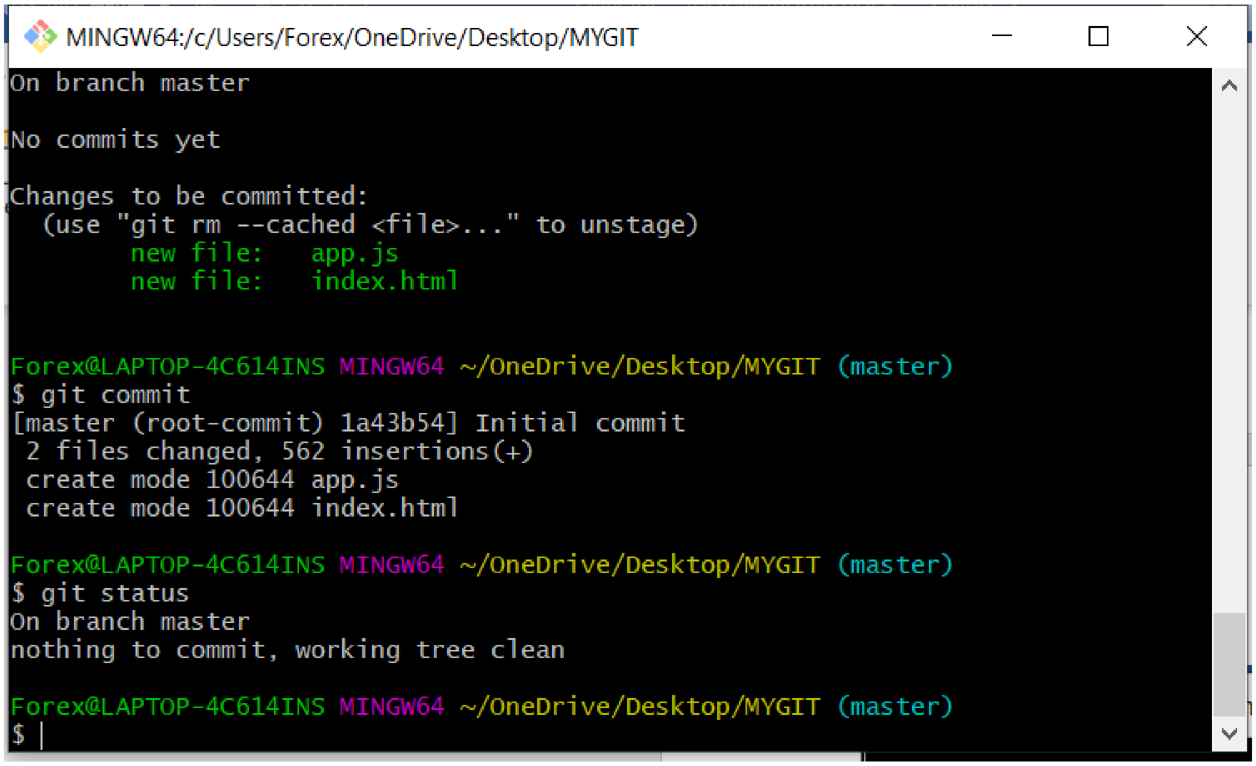
$ git commit



MINGW64:/c/Users/Forex/OneDrive/Desktop/MYGIT

```

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
#
# Initial commit
#
# Changes to be committed:
#       new file:   app.js
#       new file:   index.html
#
~
~
~
~
~
~
~
~
~
~
<rex/OneDrive/Desktop/MYGIT/.git/COMMIT_EDITMSG [unix]  (08:33 02/10/2021)1,1 All
-- INSERT --
```

Remove the # sign to allow the system to read initial commit and save using :wq



MINGW64:/c/Users/Forex/OneDrive/Desktop/MYGIT

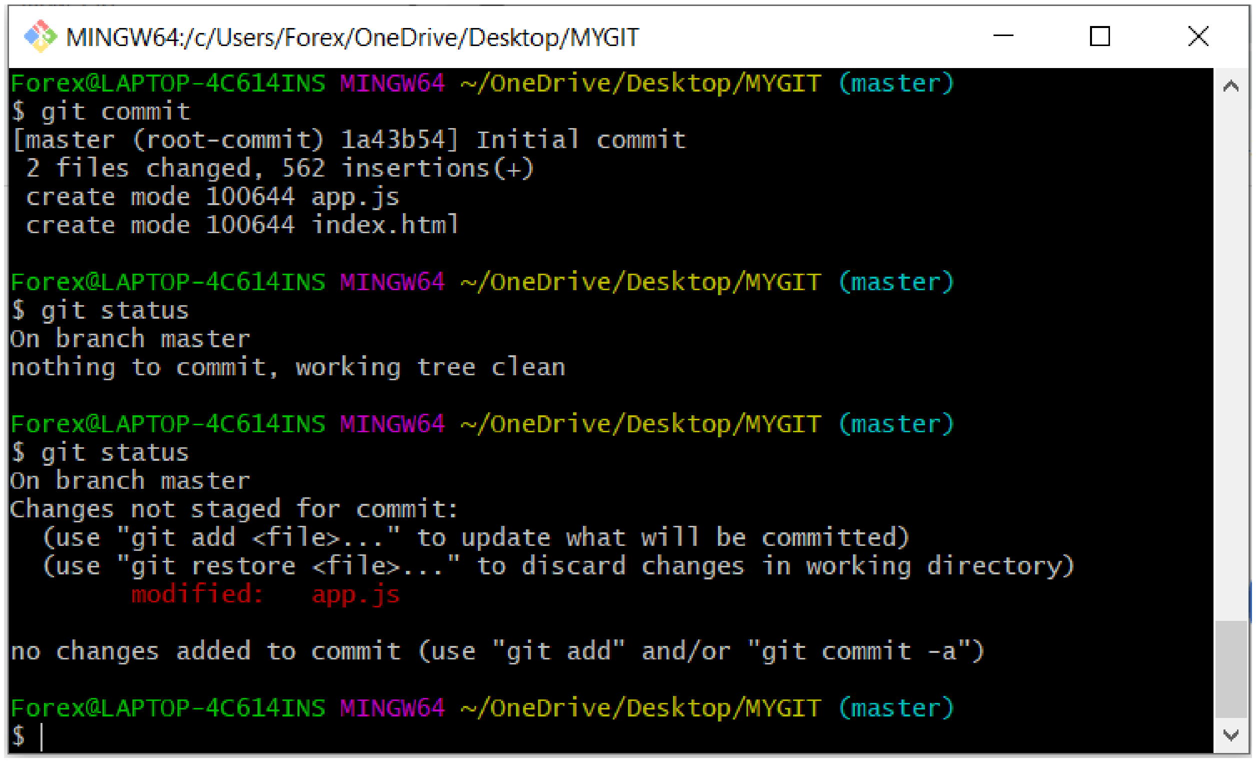
```
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   app.js
        new file:   index.html


Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git commit
[master (root-commit) 1a43b54] Initial commit
 2 files changed, 562 insertions(+)
 create mode 100644 app.js
 create mode 100644 index.html

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git status
On branch master
nothing to commit, working tree clean

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ |
```

Files have been committed on branch master and also the number is indicated

Let us edit the app.js file

Then run the command

$ git status

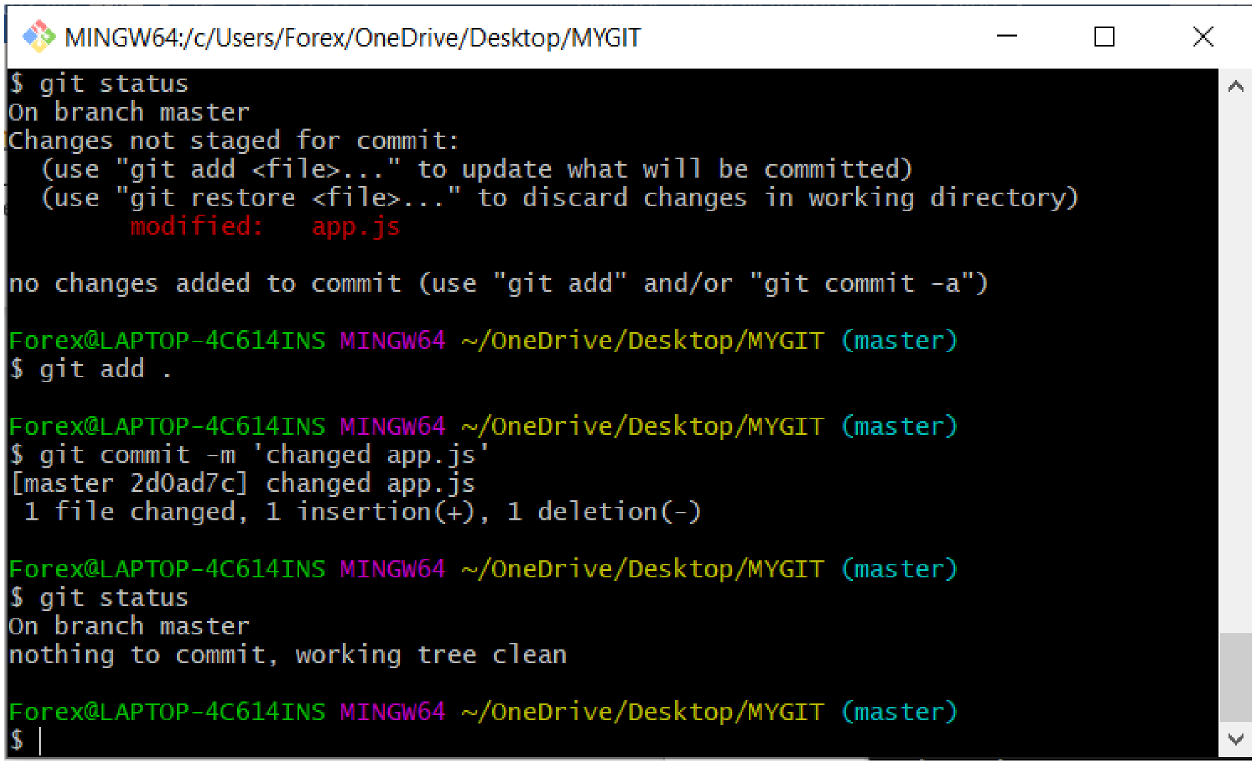


```
Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git commit
[master (root-commit) 1a43b54] Initial commit
 2 files changed, 562 insertions(+)
 create mode 100644 app.js
 create mode 100644 index.html

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git status
On branch master
nothing to commit, working tree clean

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   app.js

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

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ |
```

Run the command $ git add . to add the changes you made in the file

We can skip the editing section when committing the file by simply running the command

$ git commit -m ‘changed app.js’



```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   app.js

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

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git add .

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git commit -m 'changed app.js'
[master 2d0ad7c] changed app.js
 1 file changed, 1 insertion(+), 1 deletion(-)

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git status
On branch master
nothing to commit, working tree clean

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ |
```

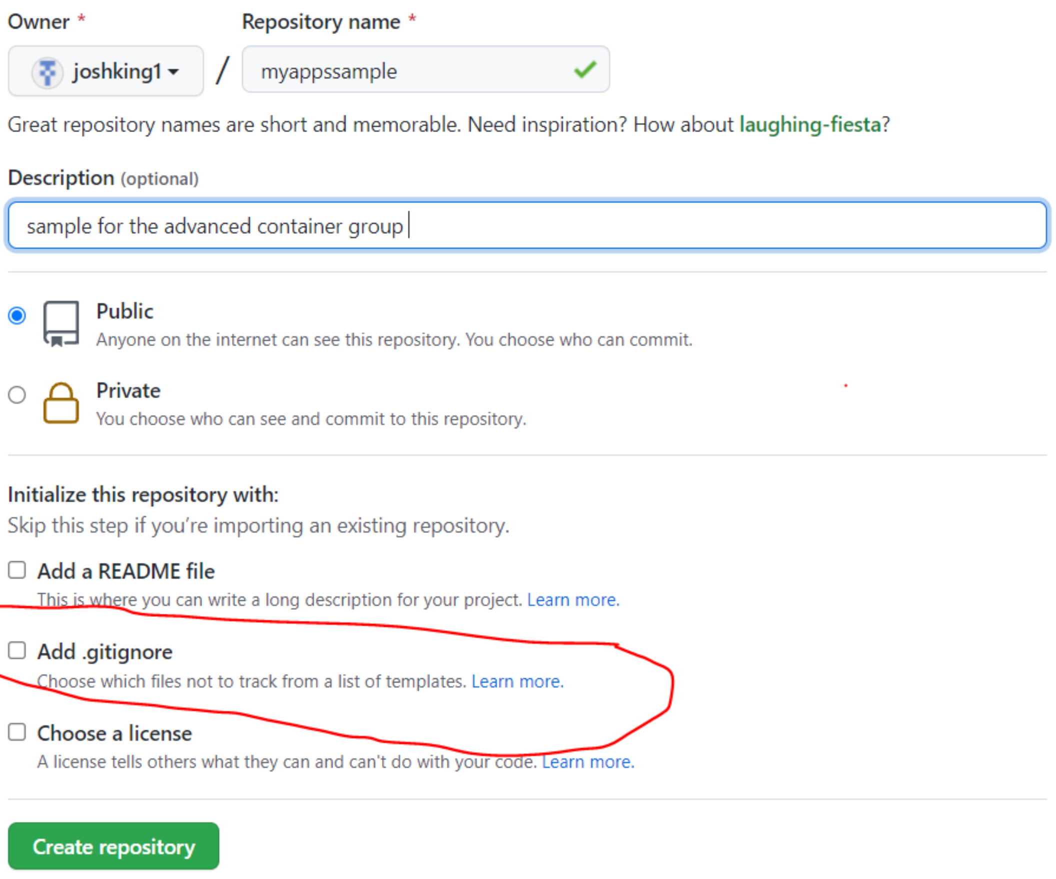
It has skipped the whole editing process and committed the file

You can confirm this by running the command $ git status

How to ignore files in the repository

This is vital when using the command

$ git add . which is going to add all the files



Owner * joshking1 / Repository name * myappssample

Great repository names are short and memorable. Need inspiration? How about laughing-fiesta?

Description (optional)

sample for the advanced container group

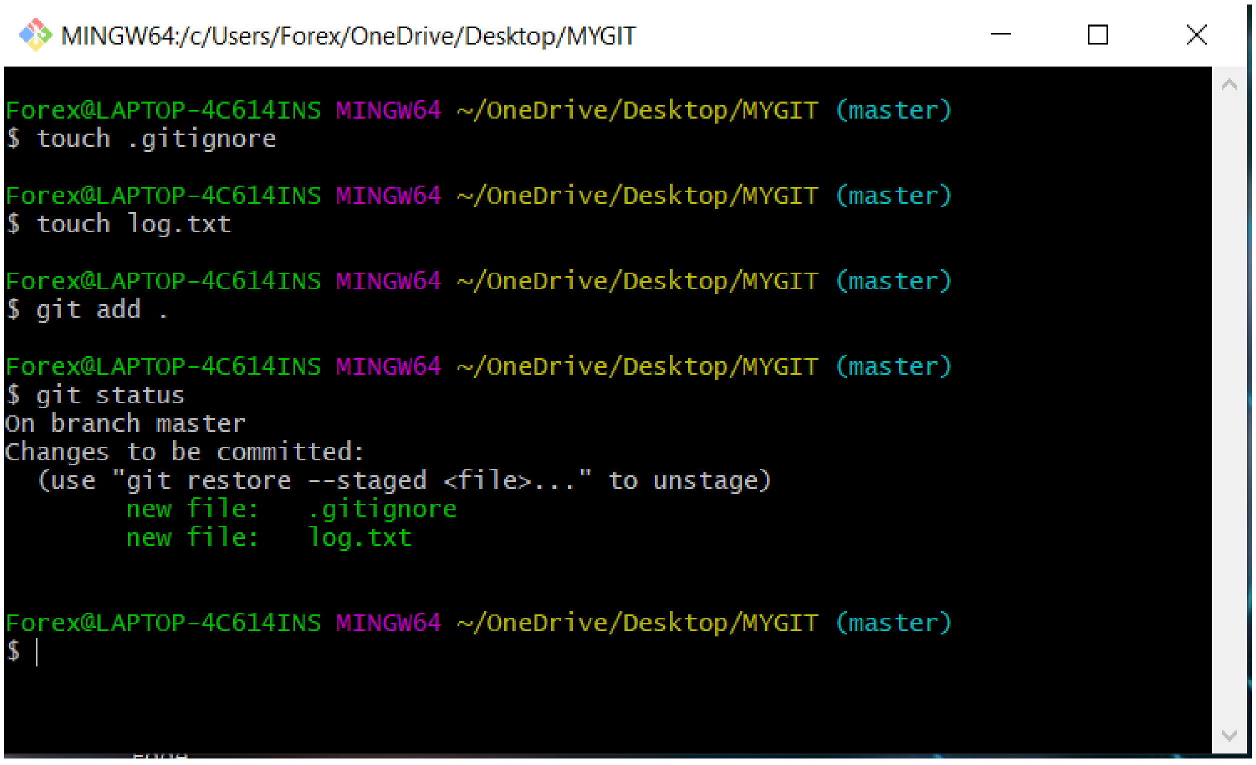
[✓ Public] Anyone on the internet can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

- [ ] Add a README file
  This is where you can write a long description for your project. Learn more.
- [ ] Add .gitignore
  Choose which files not to track from a list of templates. Learn more.
- [ ] Choose a license
  A license tells others what they can and can't do with your code. Learn more.

Create repository



MINGW64:/c/Users/Forex/OneDrive/Desktop/MYGIT

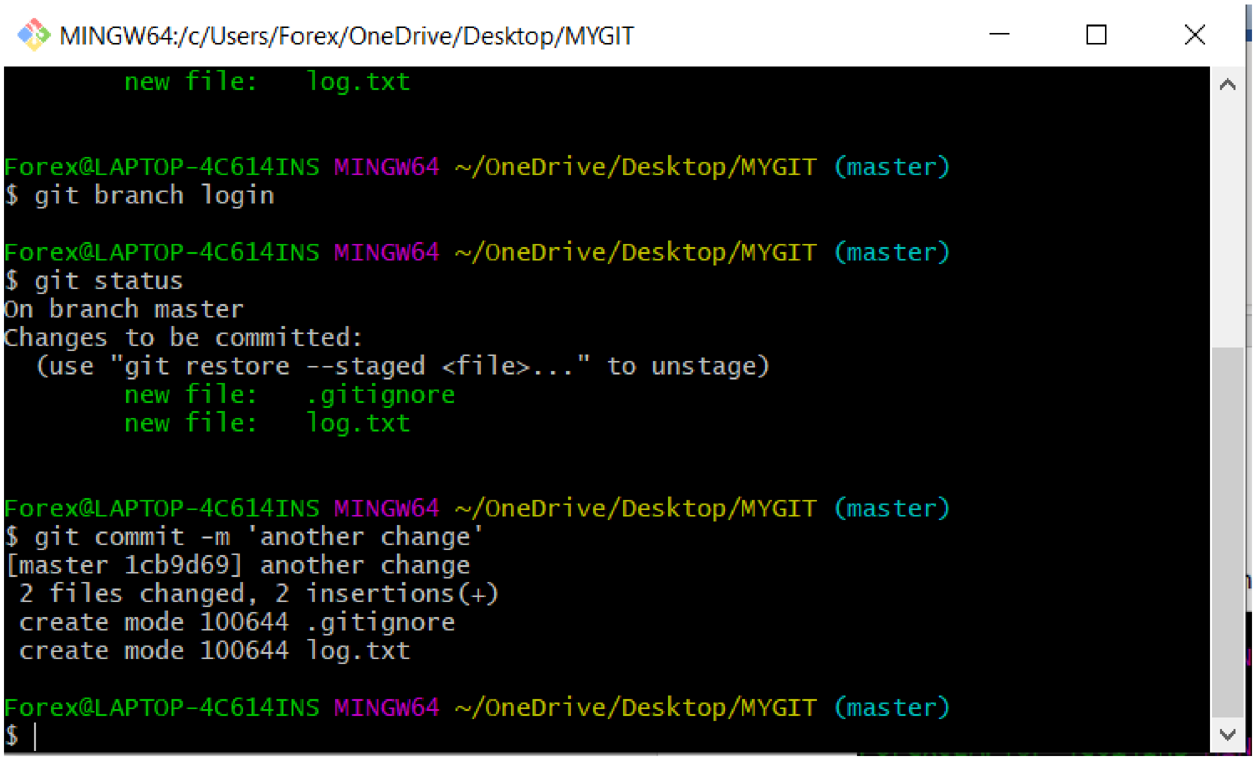
```
Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ touch .gitignore

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ touch log.txt

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git add .

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .gitignore
        new file:   log.txt


Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ 
```

You can see the .gitignore file is ignored

Log.txt is also ignored because we added it into the .gitignore file

Commit all the files

$ git commit -m ‘another change’ - you can skip editing part with this command



MINGW64:/c/Users/Forex/OneDrive/Desktop/MYGIT

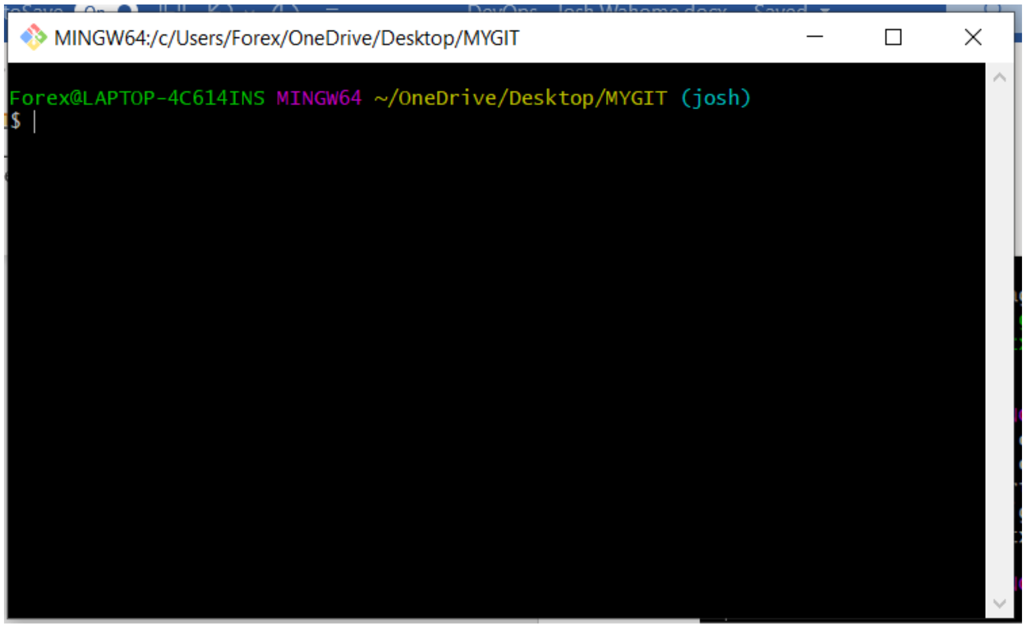
```
        new file:   log.txt


Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git branch login

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .gitignore
        new file:   log.txt


Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git commit -m 'another change'
[master 1cb9d69] another change
 2 files changed, 2 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 log.txt

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ |
```

How to create a branch in git

$ git branch josh

To switch to this branch, run the command

$ git checkout



```
MINGW64:/c/Users/Forex/OneDrive/Desktop/MYGIT

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (josh)
$ |
```

We are now in the new branch

Let create a new file

$ touch login.form

Then commit the file

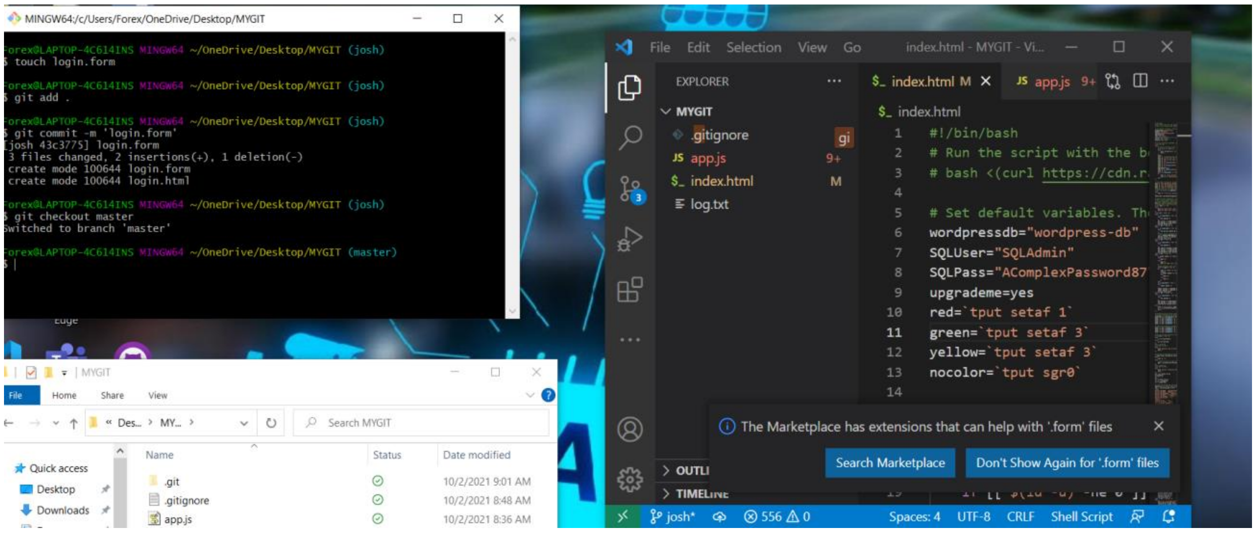
$ git commit -m ‘login.form’

Let switch to the master

$ git checkout master

Watch what happens to the login.form in the editor

It disappears



```
Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (josh)
$ touch login.form

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (josh)
$ git add .

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (josh)
$ git commit -m 'login.form'
[josh 43c3775] login.form
 3 files changed, 2 insertions(+), 1 deletion(-)
 create mode 100644 login.form
 create mode 100644 login.html

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (josh)
$ git checkout master
Switched to branch 'master'

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ |
```

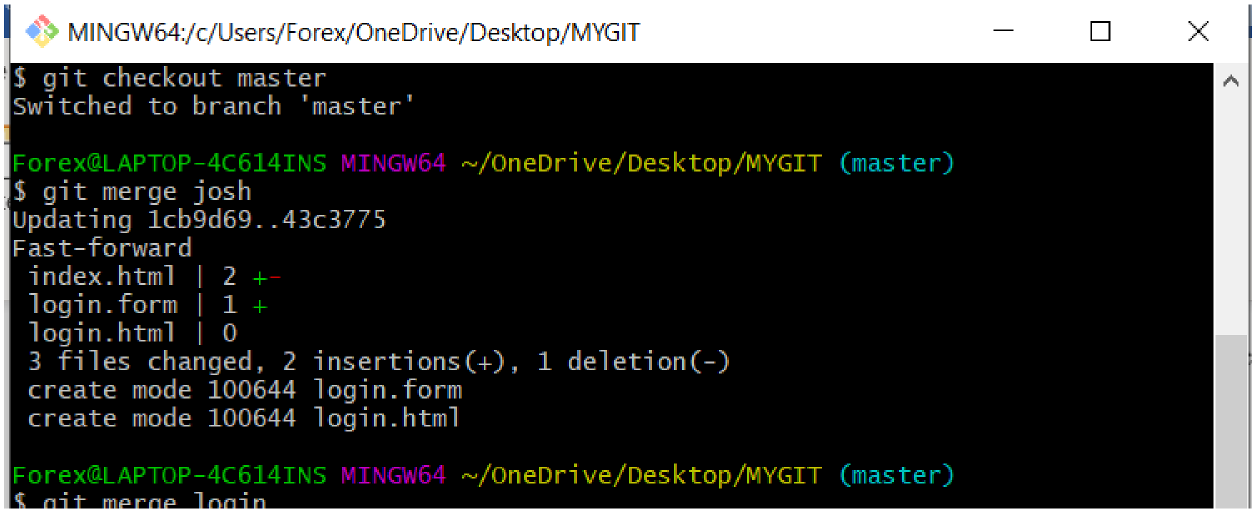
```
#!/bin/bash
# Run the script with the b
# bash <(curl https://cdn.r

# Set default variables. Th
wordpressdb="wordpress-db"
SQLUser="SQLAdmin"
SQLPass="AComplexPassword87
upgrademe=yes
red=`tput setaf 1`
green=`tput setaf 3`
yellow=`tput setaf 3`
nocolor=`tput sgr0`
```

The Marketplace has extensions that can help with '.form' files
Search Marketplace
Don't Show Again for '.form' files

This is because it is not part of the master branch but josh branch

You can merge the files by running the command

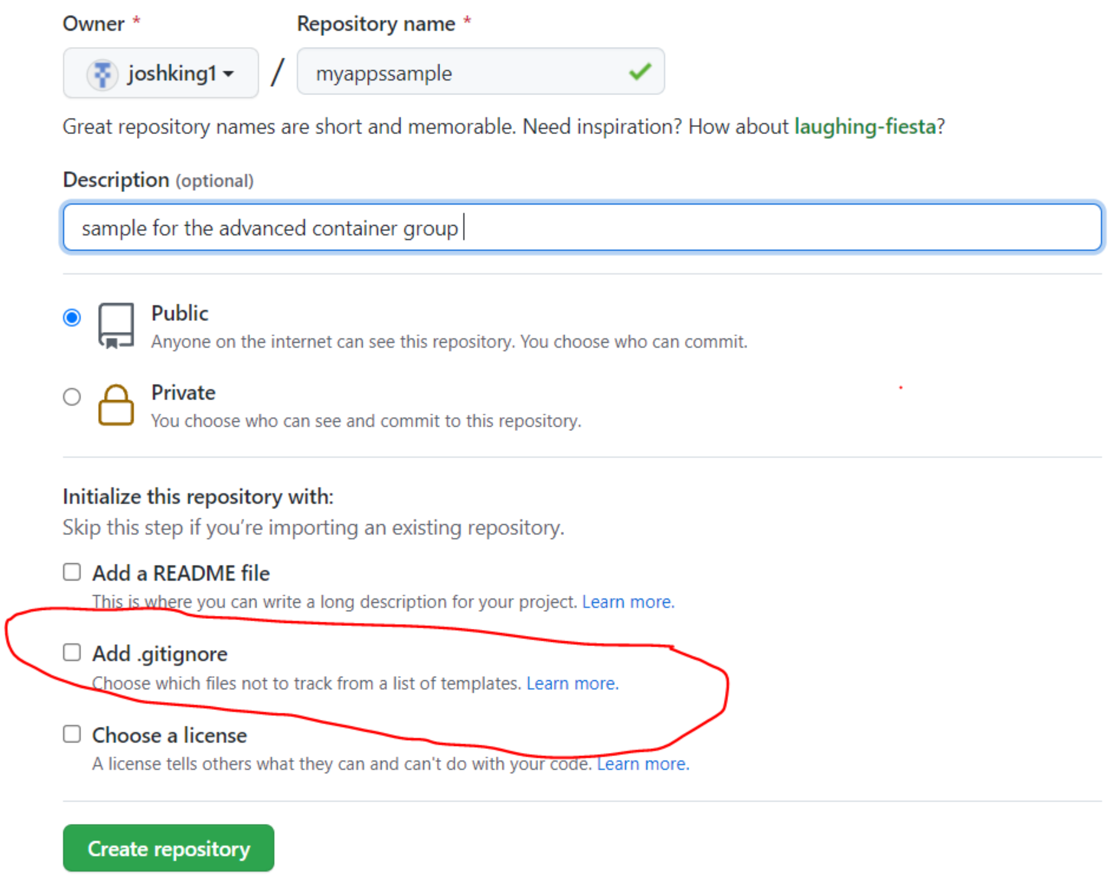
$ git merge josh



```
$ git checkout master
Switched to branch 'master'

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git merge josh
Updating 1cb9d69..43c3775
Fast-forward
 index.html | 2 +-
 login.form | 1 +
 login.html | 0
 3 files changed, 2 insertions(+), 1 deletion(-)
 create mode 100644 login.form
 create mode 100644 login.html

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git merge login
```

Let now jump to Github and wok with repository

GITHUB



Owner * joshking1 / Repository name * myappssample

Great repository names are short and memorable. Need inspiration? How about laughing-fiesta?

Description (optional): sample for the advanced container group

[✓ Public] Anyone on the internet can see this repository. You choose who can commit.

Private — You choose who can see and commit to this repository.

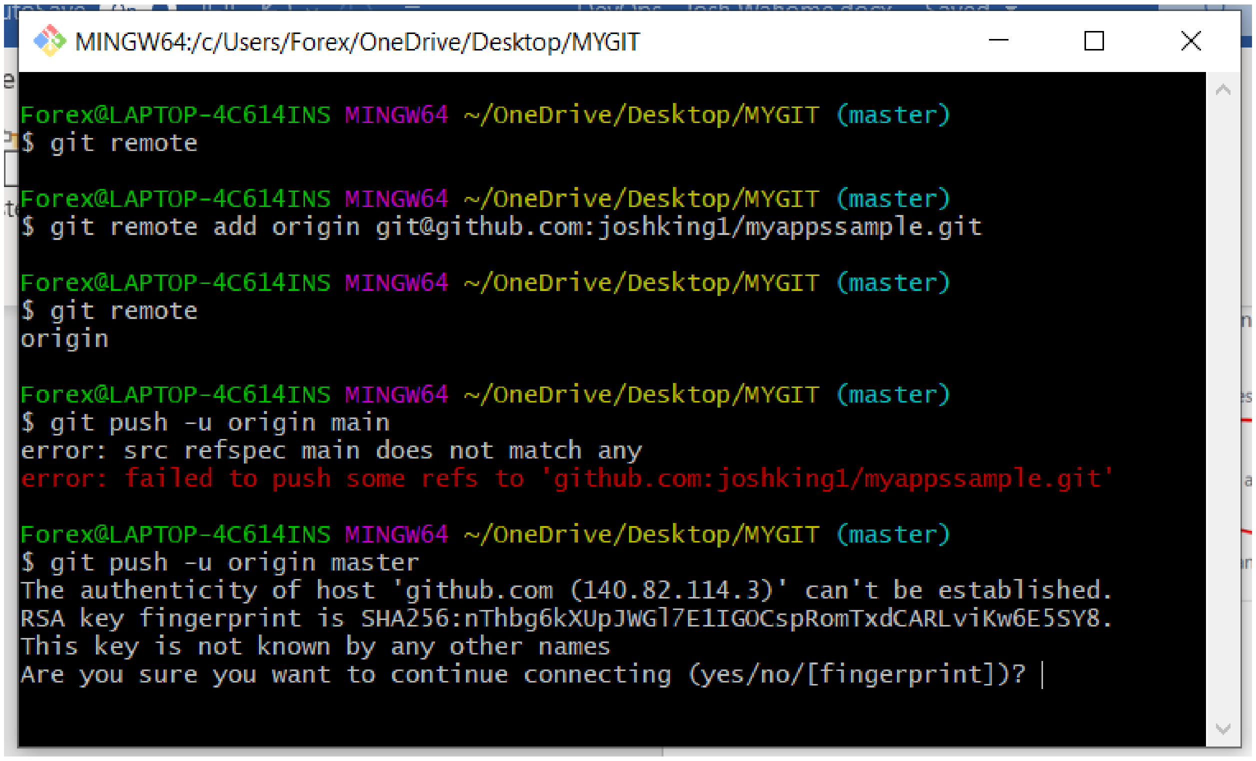
Initialize this repository with:
Skip this step if you're importing an existing repository.

- [ ] Add a README file — This is where you can write a long description for your project. Learn more.
- [ ] Add .gitignore — Choose which files not to track from a list of templates. Learn more.
- [ ] Choose a license — A license tells others what they can and can't do with your code. Learn more.

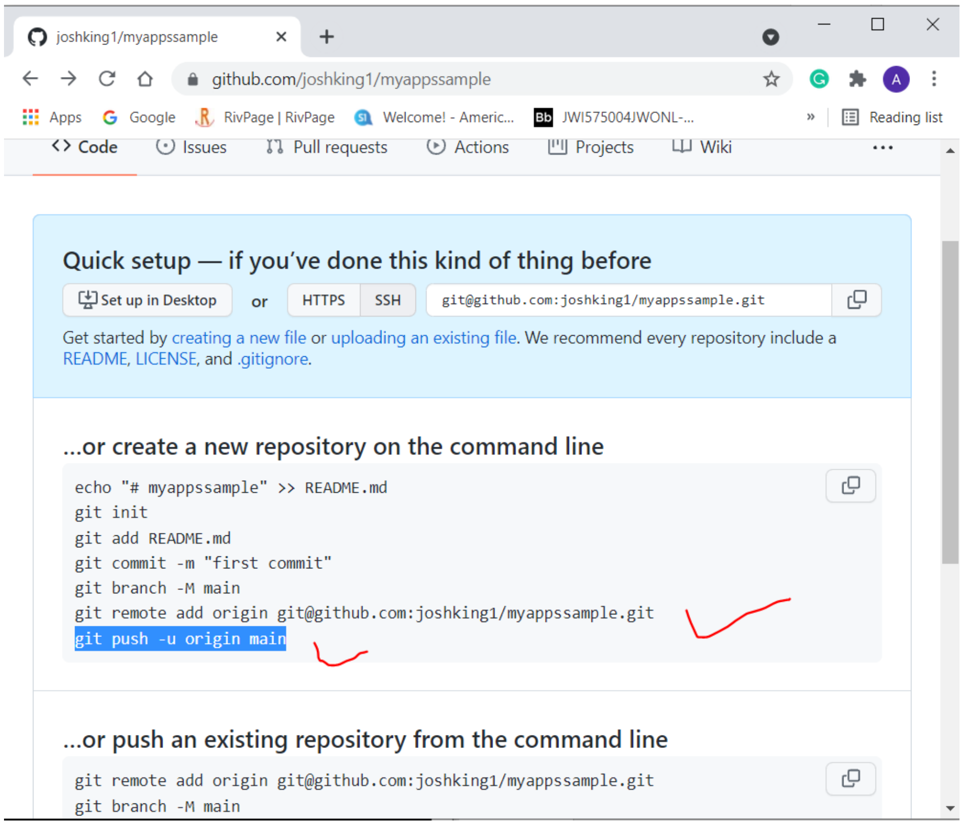
Create repository

Run the command

$ git remote



```
Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git remote

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git remote add origin git@github.com:joshking1/myappssample.git

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git remote
origin

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git push -u origin main
error: src refspec main does not match any
error: failed to push some refs to 'github.com:joshking1/myappssample.git'

Forex@LAPTOP-4C614INS MINGW64 ~/OneDrive/Desktop/MYGIT (master)
$ git push -u origin master
The authenticity of host 'github.com (140.82.114.3)' can't be established.
RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```


Quick setup — if you've done this kind of thing before

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# myappssample" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:joshking1/myappssample.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin git@github.com:joshking1/myappssample.git
git branch -M main
```

$ git remote add origin git@github.com:joshking1/myappssample.git

$ git remote

Should say origin

$ git push -u origin main

Should display an error

$ git push -u origin master

(be careful here – should be master not main)