In the previous lesson we looked a git and what we as developers can use version control for. We also set up an SSH key and added it to the SSH agent on our local machines and added the key to our GitHub accounts, to handle authorization when we pull and push code to and from the repository.

\*\*\_Knowledge check: What are the 3 stages?\_\*\*

Let's now take a look at Github:

### Setting Up Github SSH Keys For Authorisation and Access

Firstly, we need to create an account in the General Assembly github enterprise. Go to [the invite page](https://git-invite.generalassemb.ly/invite) and create an account.

After setting up an account, we need to configure our access privileges. During the course we are going to be using SSH to access our repositories so lets set that up now.

1. Open the terminal on VM and run the command `ssh-keygen -t ed25519 -C "your\_email@example.com"`

this will generate a new SSH key on your VM, using the provided email as a label.

At the prompt, press enter. We don't need to worry about adding passphrases so just hit enter until the SSH key is created.

2. Navigate to `~/.ssh` by typing the command cd ~/.ssh

3. Create a config file `touch config`

4. Open the file in VSCode with `code config`

5. We need to update our config file so paste in this code snippet:

```

Host \*

AddKeysToAgent yes

IdentityFile ~/.ssh/id\_ed25519

```

6. Run `

`

7. Open the ssh key file `code ~/.ssh/id\_ed25519.pub` and copy the contents

8. Go to your github enterprise account and select settings from the account menu in the top right.

9. Go to SSH and GPG keys and add a new SSH key. Give it a title and paste the key into the box.

You're done!

## Guided Practice: Making and cloning repositories (10 mins)

Let's do this together:

1. Go to your GitHub account

2. In the top left, hit the `+` button and select `New repository`

![](https://help.github.com/assets/images/help/repository/repo-create.png)

3. Name your repository `hello-world`

![](https://help.github.com/assets/images/help/repository/repo-create-name.png)

4. \*\*Initialize this repository with a README\*\* (So that we can `git pull`)

5. Click the big **green** Create Repository button

We now need to connect our local Git repo with our remote repository on GitHub. We have to add a "remote" repository, an address where we can send our local files to be stored.

```bash

git remote add origin git@github.com:github-name/hello-world.git

```

You can find this address by clicking on the "Clone or download" button:

<img src="https://i.imgur.com/PNyQ2YW.png" align="center">

#### Pushing to GitHub

In order to send files from our local machine to our remote repository on GitHub, we need to use the command `git push`. However, you also need to add the name of the remote, in this case we called it `origin` and the name of the branch, or "copy" of the code, in this case `master`. (Don't worry about branches just yet!)

```bashmaster

git push origin master

```

In other words: "Hey Git, take the code I have locally and send it to the 'master' copy located at the 'origin' web address".

This should fail due to new files on the remote repo.

> Check: Did students get a failure message?

#### Pulling from GitHub

As we added the README.md in our repo, we need to first `pull` that file to our local repository to check that we haven't got a 'conflict', or a difference between our remote and local code.

```bash

git pull origin master

```

In other words: "Hey Git, give me the current code from the 'master' copy located at the 'origin' web address".

Once we have done this, you should see the README file on your computer. Now you can push your changes:

```bash

git push origin master

```

Refresh your GitHub webpage, and the files should be there.

> Check: Were students able to get their code on GitHub?

#### Cloning your first repository

Now that everyone has their first repository on GitHub, let's clone our first repository!

Cloning allows you to get a local copy of a remote repository.

Navigate back to your Desktop and \*\*delete your hello-world repository\*\*:

```bash

cd ~/Desktop

rm -rf hello-world

```

Go to your github account and copy the address. Paste it into the chat and when everybody has pasted theirs in, copy the one below yours. If you're the last in the list, take the one from the top of the list.

<img src="https://i.imgur.com/mnnEwUN.png" align="center">

Ensure that you have SSH checked and copy this url.

#### Clone their repo!

To retrieve the contents of their repo, all you need to do is:

```bash

$ git clone git@github.com:username/hello-world.git

```

Git should reply:

```bash

Cloning into 'hello-world'...

remote: Counting objects: 3, done.

remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0

Receiving objects: 100% (3/3), done.

Checking connectivity... done.

```

> Check: Did students get this message?

You should have another 'hello-world' folder on your machine - `ls` to check it out. You now have cloned your first repository!

## Introduction: What is forking? (5 mins)

The `fork` and `pull` model lets anyone fork an existing repository and push changes to their personal fork without requiring access be granted to the source repository.

Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea.

> Check: Could someone explain the difference between forking and cloning?

#### Cloning vs Forking

When you fork a repository, you make a new \*\*remote\*\* repository that is exactly the same as the original, except you are the owner. You can then `clone` your new fork and `push` and `pull` to it without needing any special permissions.

When you clone a repository, unless you have been added as a contributor, you will not be able to push your changes to the original remote repository because it's not your GitHub repository!

#### Pull requests

When you want to propose a change to a repository (the original project) that you have forked, you can issue a pull request. This basically is you saying:

\_"I've made some changes to your repository, if you want to include them in your original one then you can pull them from my fork!"\_

We'll give this a shot in the next section.

## Independent Practice: Assess (10 mins)

Use the internet and what you've learned today to answer the following questions with a partner:

- How do I send changes to the staging area?

- How do I check what is going to be committed?

- How do I send the commits to GitHub?

- How do I go back to the previous commit?

- How do I check the configuration on a specific machine?

- How does GitHub know that I am allowed to push to a specific repo?

> Check: Call on various students to reveal their answers.

## Introduction: Git Ignore file (10 mins)

Before we wrap things up, let's have a chat about the .gitignore file.

When you create a new project, most integrated development environments usually generate files specific to your computer (i.e. setup files, temporary files, compiled code, etc). These kind of files should not be pushed to the remote Git repository, as they are specific to you alone and might affect other peoples' ability to use the project.

This is where the .gitignore file comes in.

The .gitignore file lists the type of files that should not be uploaded to your remote Git repo (i.e., what files to \_ignore\_).

You can put .gitignore files in your repo, so whoever clones your project will ignore unnecessary files. You can also set up your computer so you always ignore certain files for all of your projects - a "global gitignore". Let's do the latter now.

Go to [gitignore.io](https://www.gitignore.io/), a website that generates .gitignore files. Type in the types of projects you'll be working with (Rails, Android, OSX, Windows), and press \_Generate\_. Copy all of the generated text.

Okay, now open Terminal and create the .gitignore file wherever you want; I tend to run `touch ~/.gitignore`. Then, open it and paste the generated text into the file. Make sure to save it!

Now, you have to register the file with Git. In Terminal, run:

```bash

git config --global core.excludesfile ~/.gitignore

```

All of your future projects will ignore the files listed.

\_Note\_: For local .gitignore files, you don't have to register them with Git. Just put them in the root folder of your Git project.

## Git Updates - Switch and Restore (10 mins)

[take a few minutes to read about these new git commands that can replace some instances of the checkout command](https://www.banterly.net/2021/07/31/new-in-git-switch-and-restore/)

## Conclusion (5 mins)

As a developer, you'll have to use Git pretty much everyday - the learning curve is steep and all the principles of version control can be a bit blurry sometimes, so we ask students to push their homework everyday and to commit regularly during project time.

Don't be frustrated by all the new commands because we will definitely have the time to practice during this course.

- Explain the difference between forking and cloning.

- Describe the steps to initialize a Git repository and link your local repository to a GitHub remote location.