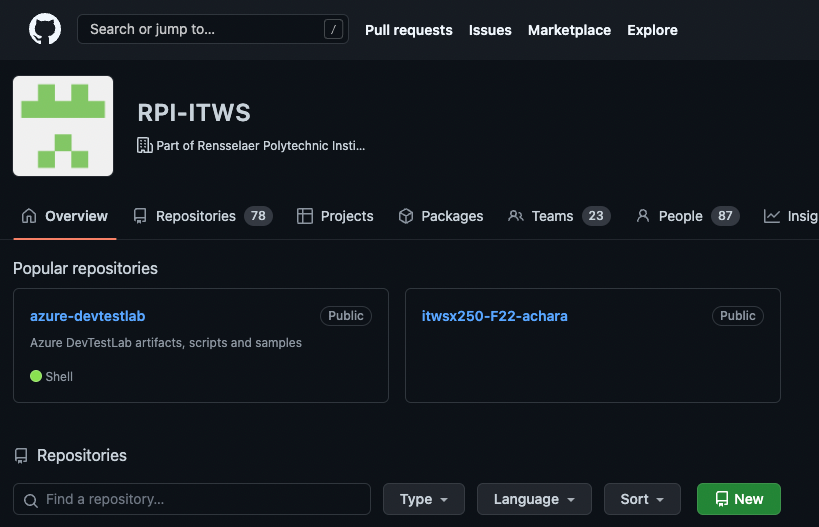
GitHub and Azure

1 – GitHub

Let’s log on to GitHub and create a repository for our classwork



Graphical user interface, text, application

Description automatically generated

Now let’s log on to our servers

2 – start the instance at portal.azure.com

Graphical user interface, application

Description automatically generated

3a – In order to have git ‘talk’ to our repository, we need to create a key pair – this will allow us to authenticate with GitHub to clone our repo.

Let’s create a key so we can authenticate to GitHub

<https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>

From a Terminal session on our local computers, SSH into your server:

ssh uid@FQDN

Now that we’re on our instances,

Let’s create a key pair using the ssh-keygen command. Type:

sudo -u www-data ssh-keygen -t ed25519 -C "[yourRCSid]@rpi.edu"

When the system asks for a filename, accept the default.

(It will then ask for a passphrase (ie password)

(Enter a password – don’t forget it!)

Verify that the file was created by typing

ls -la /var/www/.ssh

You should see at least one file with the name

id\_ed25519.pub

Now let’s start the system’s security agent service. Type:

eval "$(ssh-agent -s)"

We now need to add the *private* key to our system’s list of known users. Type

sudo -u www-data ssh-agent ssh-add /var/www/.ssh/id\_ed25519

(You will be asked for the passphrase) Type it in

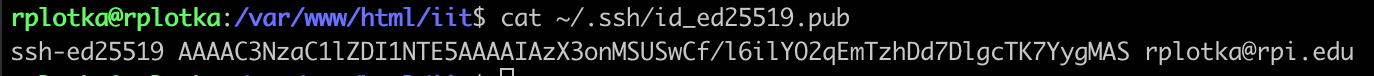
3b - Now we need to add the *public* key to GitHub

<https://docs.github.com/en/github/authenticating-to-github/adding-a-new-ssh-key-to-your-github-account>

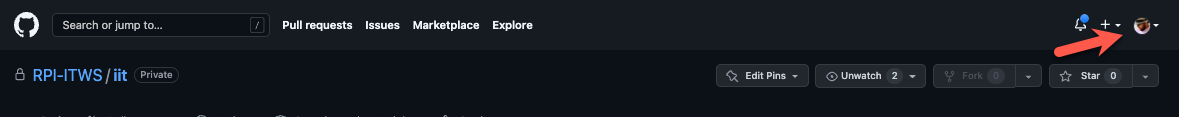
let’s copy the public key to our clipboard. Type

cat /var/www/.ssh/id\_ed25519.pub

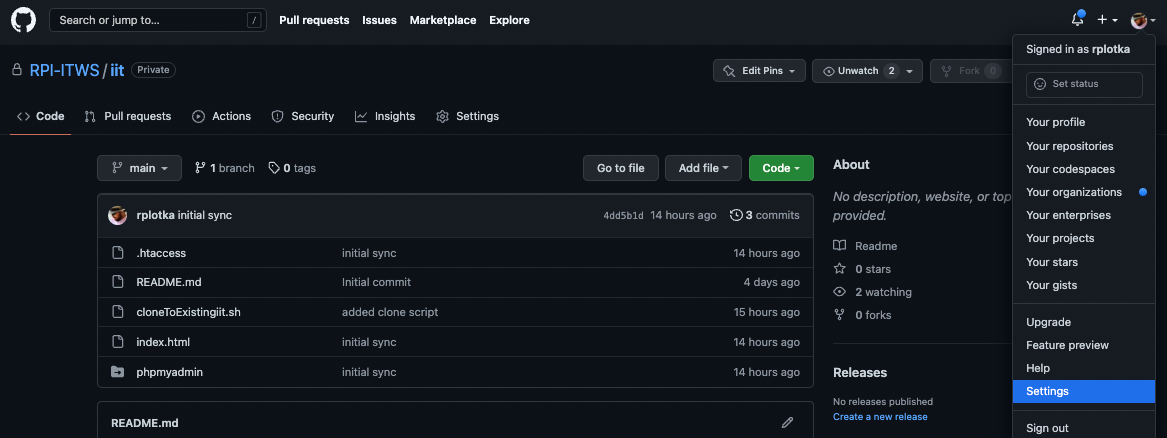
Highlight the line of text starting with ssh-ed25519… and copy it



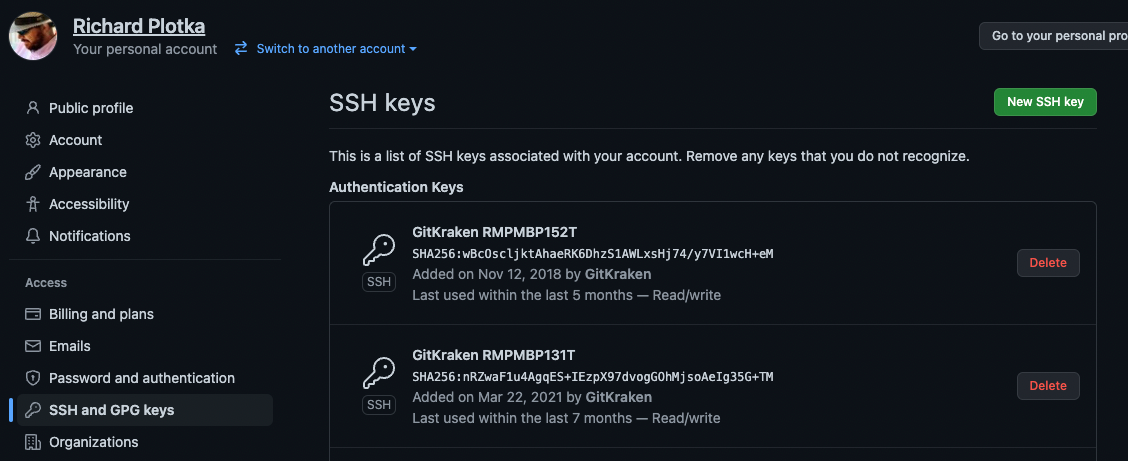
Now go to your GitHub account in your browser, and click on your profile icon



Select Settings



Select SSH and GPD keys



Click New 

Give it a name, and paste the key into the Key field

Graphical user interface, text, application

Description automatically generated

(You may be asked for your PW. If so, enter it)

4 – We have now told the git program on our Azure webservers how to ‘talk’ to GitHub. So let’s connect the repo on GitHub with the the iit folder on our servers. Let’s first clone the repo.

Let’s go back to our session on our servers, then move into your iit folder. Type

cd /var/www/html/iit

Check to see where our directory pointer is located. Type

pwd

Text

Description automatically generated

OK, let’s execute the ‘git clone’ command to connect our folder to our repo.

Type the following, replacing [yourRepoName] with the name of your repo (ITWS1100-[yourRCSid]). Type the following commands:

We want to connect our repo and our iit folder. We have a small problem – GitHub will not clone into an existing folder…So, we are going to trick it with 4 commands.

NOTE: for #1, we must use the user www-data, since our key to GitHub uses this ID. For 2-4, you should not need the www-data user parameter. They ‘should’ only be needed when issuing a command to your GitHub repo.

1. Clone the repo into a temporary folder named tmp
   1. sudo -u www-data git clone git@github.com:RPI-ITWS/[yourRepoName].git tmp
2. Move (mv=move) the .git folder (which contains the history that sits in GitHub) into our iit folder
   1. sudo mv tmp/.git .
3. Delete (rm=remove) the temporary folder
   1. sudo rm -rf tmp
4. Reset gits pointer to compare git in our iit folder with the ‘known’ inventory on GitHub
   1. sudo -u www-data git reset

Might need to enter if told by linux

git config –global –add safe.directory /var/www/html/iit

Let’s see what happened by checking the status of git on our machines. type

git status

(You will see something similar to the below because we have the files we already created in Lab 1, and the README.md file we just created when we created our repos)

Text

Description automatically generated

This tells us that we have files on GitHub that are not on our server and files on our server that are not on GitHub.

We want to get both sides synchronized.

So:

The system thinks we deleted files locally – so let’s restore the index, so let’s put them into the inventory. Type

sudo -u www-data git restore .

Let’s copy (pull) the files down from the online repo

sudo -u www-data git pull

Let’s check our folder. Type

ls -la

Text

Description automatically generated

We can see that we now have both the files from the repo and our local files from before on our system. However, git does not yet know that we want it to track our local files, so…

Let’s add our local files. Type

sudo -u www-data git add .

Now check git. Type

git status

Text

Description automatically generated

This tells us that we are tracking everything and that our new (files not yet sent to the server) files are waiting to be committed to the local repo

So, let’s commit them into the local repo. (the -m flag allows us to put in the required commit message right from the command line) Type

NOTE: If you cut/paste the command below – you will need to retype the single quotes

sudo -u www-data git commit -m ‘initial sync’

and

git status

Text

Description automatically generated

This tells us that our local repo is up to date but that we are out of sync ‘ahead’ of (more current than) our online version – so let’s fix that by sending our local repo up to GitHub and updating our online repo

Send the local files to GitHub. Type

sudo -u www-data git push

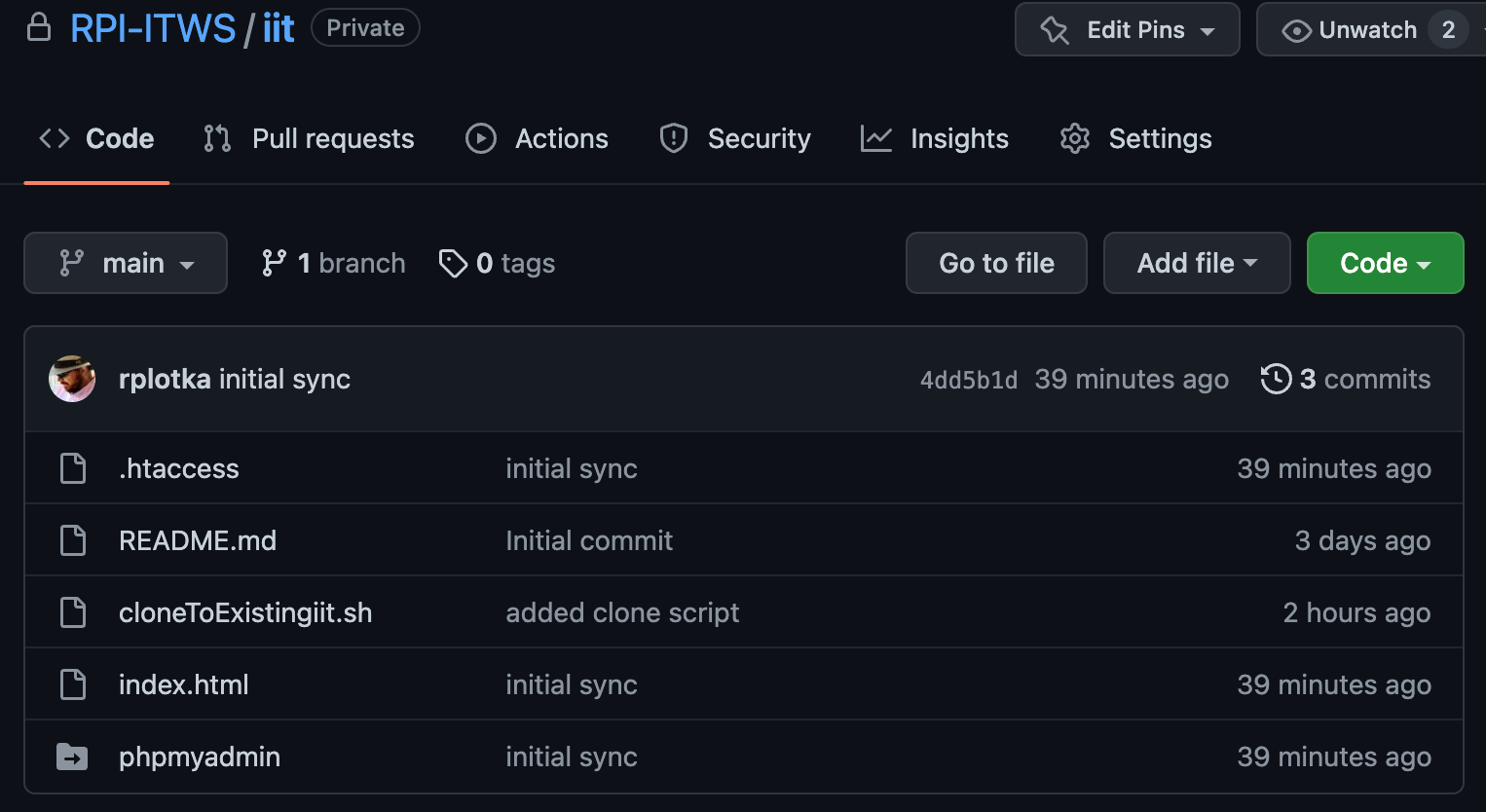
and

git status

**Text

Description automatically generated**

Let’s check our GitHub repo in our bowsers



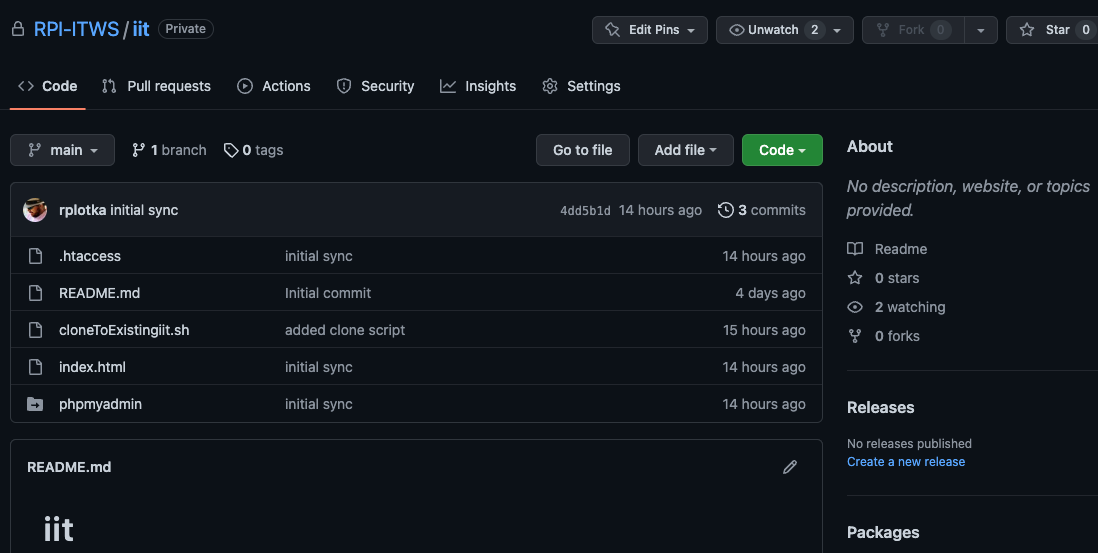
Our GitHub repo and Web Server’s iit folder are now connected and synced! Well Done!

We now need to connect our GitHub repos to our local machines for development. This will be much easier since we will start fresh.

5 – Clone the repo to our local machine

Using our Browser – eg Chrome, lets open our GitHub Repo

(In this example, my repo is called iit – yours should be called itws1100-[yourRCSid])



Click the  button

Select ‘Open with GitHub Desktop

Graphical user interface, application

Description automatically generated

Github Desktop will open

(You may get a prompt asking for permission – allow it)

You need to select a folder on your local machine for our class work. In the example here, I have a folder called ITWS1100-ClassFiles. I am going to have GitHub put my repository there and name it iit (for clarity) you may use any name you like, including the default.

Graphical user interface, application

Description automatically generated

Click the ‘Clone’ button

GitHub Desktop will create the folder and copy the files into it.

When done, GitHub Desktop will give you a screen allowing you to do a few things.

Graphical user interface, application

Description automatically generated

If you click the View the Files link – your system’s file explorer will open to that folder/directory

If you click ‘Open with Visual Studio Code’, VS Code will open with all of your files

But first – we need to copy our lab 1, and lab 2 folders and make a lab 3 folder

Using our normal File Explorer (Finder on Mac), lets copy our lab1, and lab 2 folders into the new iit folder, and create a lab 3 folder if you don’t already have one.

Graphical user interface, text, application

Description automatically generated

Now let’s check out GitHub Desktop – notice the screen has changed, reflecting all our changes on the left. (just like a git status from the command line)

Graphical user interface, text

Description automatically generated

Let’s commit these changes. Type a message into the Summary field (remember the -m message before?). Then enter some detail in the Description field – something meaningful to remind you why you are doing this. Click Commit

Graphical user interface, text

Description automatically generated

After clicking commit, we get the following screen

Graphical user interface, application

Description automatically generated

Click Push Origin – this sends our local repo to GitHub. Exactly like the ‘git push’ from the command line above

Check your results on GitHub in the Browser – you will see all of your files….

A screenshot of a computer

Description automatically generated with medium confidence

Now – lets go put these onto our web servers…

7 – Lets bring our changes down from GitHub to our Cloud Servers on Azure

(Note: I am going to start giving you instructions and letting you determine the appropriate command)

Make sure your server is running:

SSH into your server

CD into your iit folder

Let’s ‘fetch’ the current status of our repo, type

sudo -u www-data git fetch

Text

Description automatically generated

Now let’s check the status

git status

Text

Description automatically generated

Let’s bring down our changes, type

sudo -u www-data git pull

Graphical user interface

Description automatically generated

check to make sure it worked

git status

and then

ls -la

Graphical user interface, text

Description automatically generated

OK – Now everything is synced!!!

Check it out..

You have now setup a GitHub repo and connected it with a working folder on your local computer for editing and with your servers for deployment.

Going forward, we will work on our local machines, commit and push our changes to our Repos and then pull them down to our Azure Web Server Instances to deploy/publish.

Welcome to DevOps!

Final Note: Notice that the procedures we follow from the command line on our webservers are the same as the ones we follow through the interface in GitHub Desktop. GitHub Desktop is just showing us prettier screens and executing the appropriate command for us when we click the pretty buttons. If you open a Terminal session on your local machines and navigate (set your file pointer using CD) to your iit folder, and type a git status, you will see something familiar.