Version Control with Git

BY ANGELA XUE, SNAP

Taken from https://swcarpentry.github.io/git-novice/

Part 0: Set-up

Step 1: Install Git

Step 2: Create GitHub Account

Step 3: Open Terminal Application

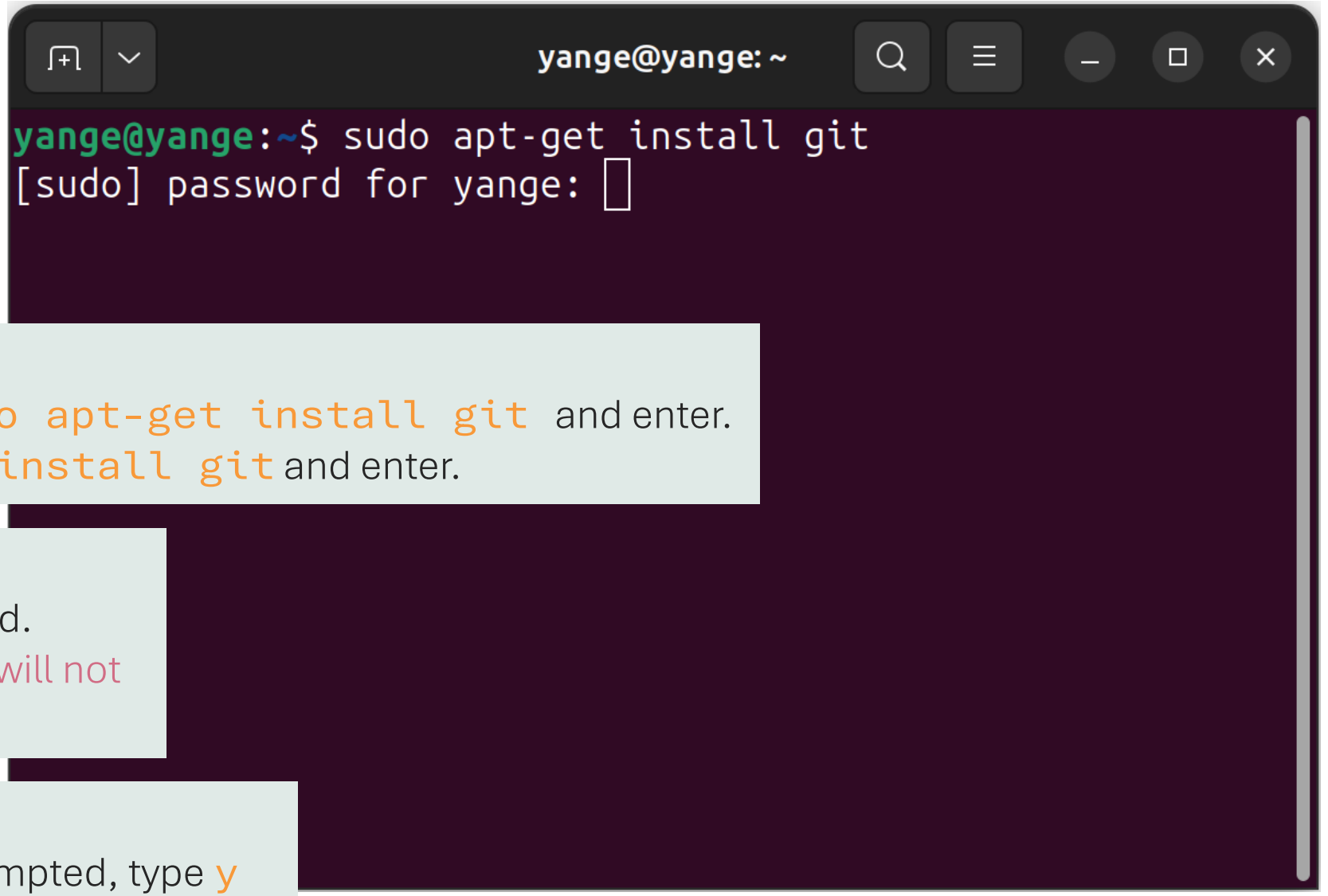
Installing Git

Navigate to your OS for instructions.

Instructions from the Carpentries:
https://carpentries.github.io/workshop-template/install_instructions/#git

Linux

1:
Press “ctrl+alt+T” simultaneously to open the terminal.



2:
For Ubuntu/Debian, type `sudo apt-get install git` and enter.
For Fedora, type `sudo dnf install git` and enter.

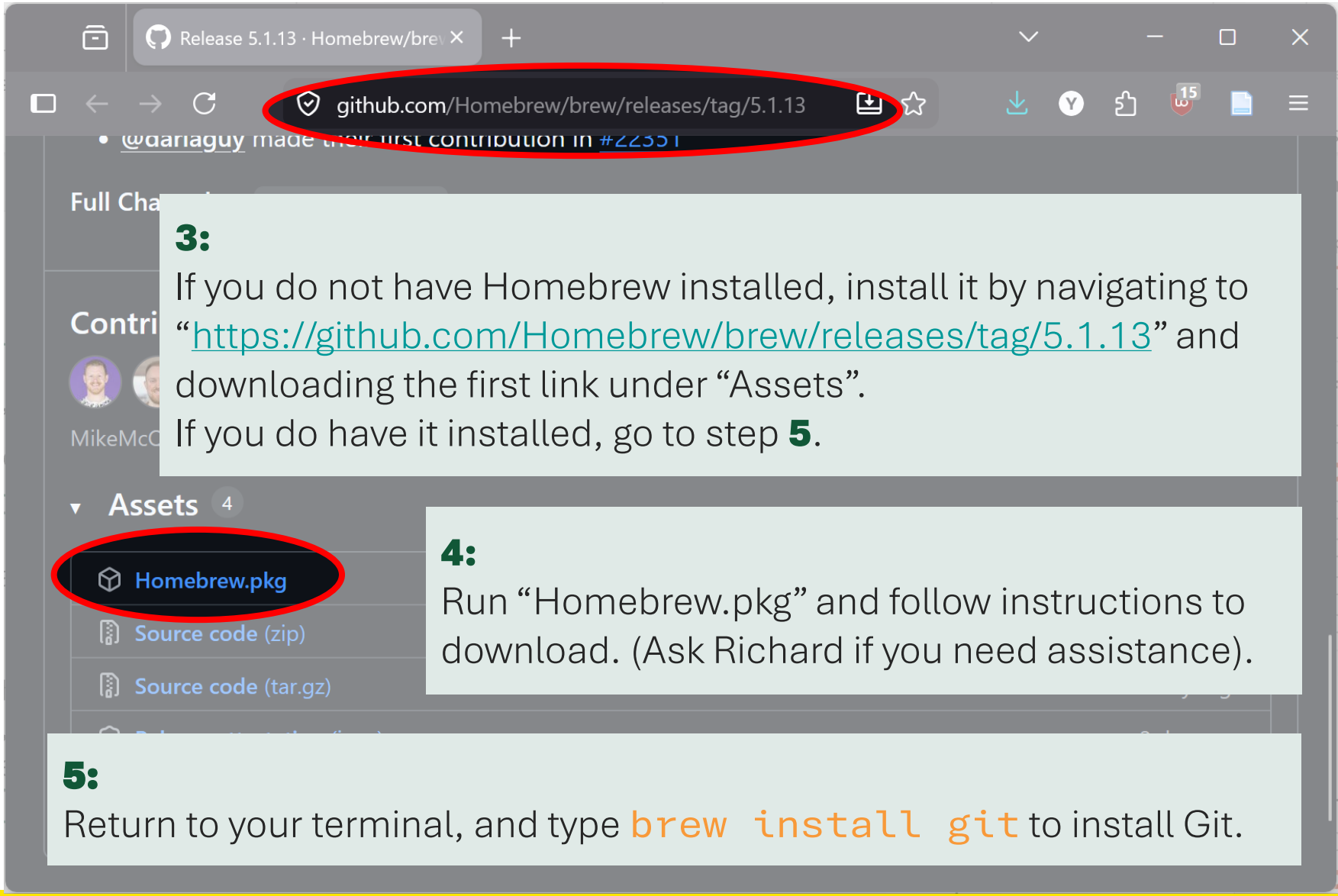
3:
Enter in your password.
Note: your password will not appear as you type it.

4:
When prompted, type y and enter to install Git.

MacOS

1:
Type “terminal” into the search bar and enter.

2:
Check if git is already installed by typing `git -version`.
If it is installed, the terminal should show the version number. If not, it will show an error. Continue to step **3**.

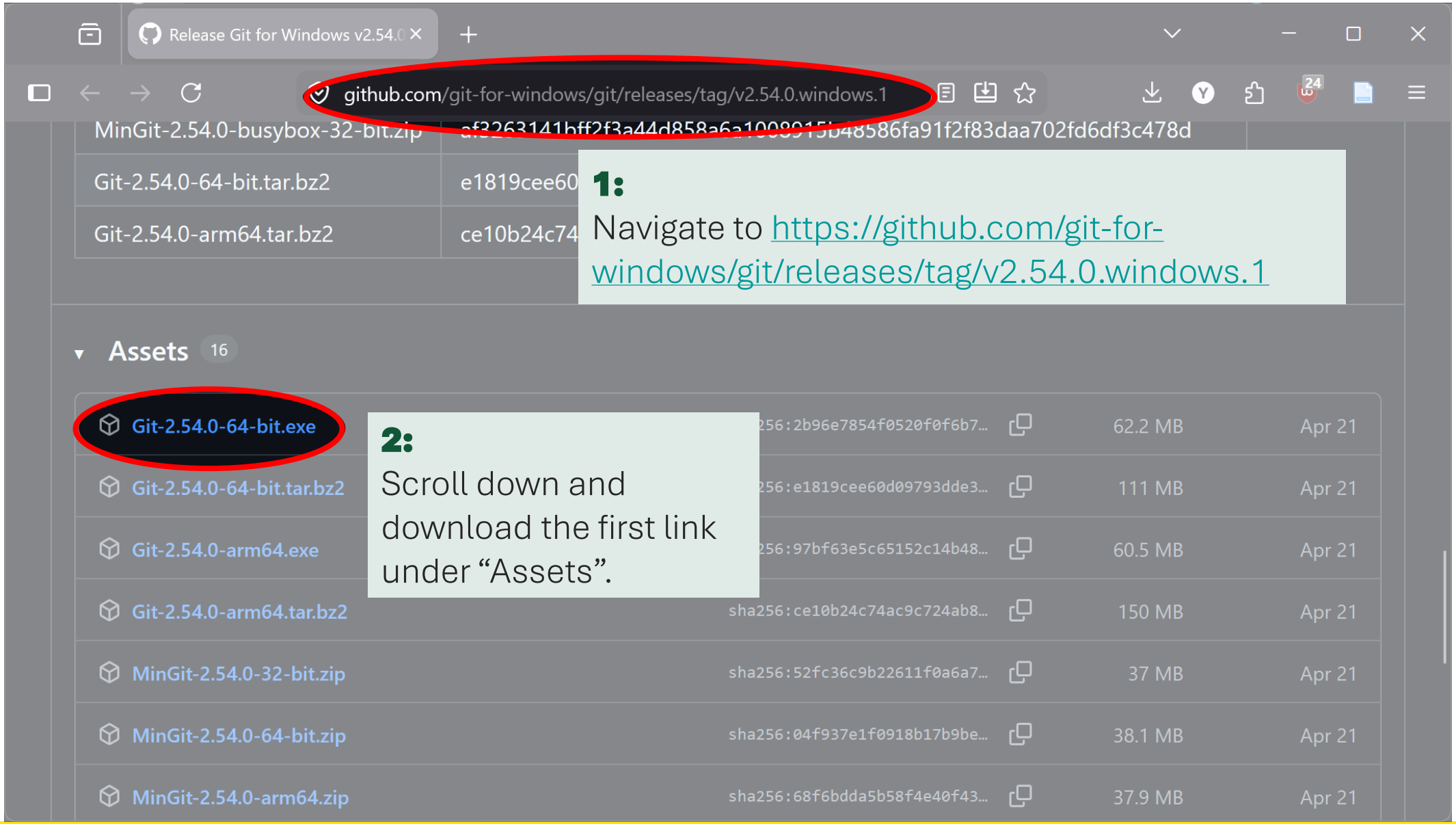


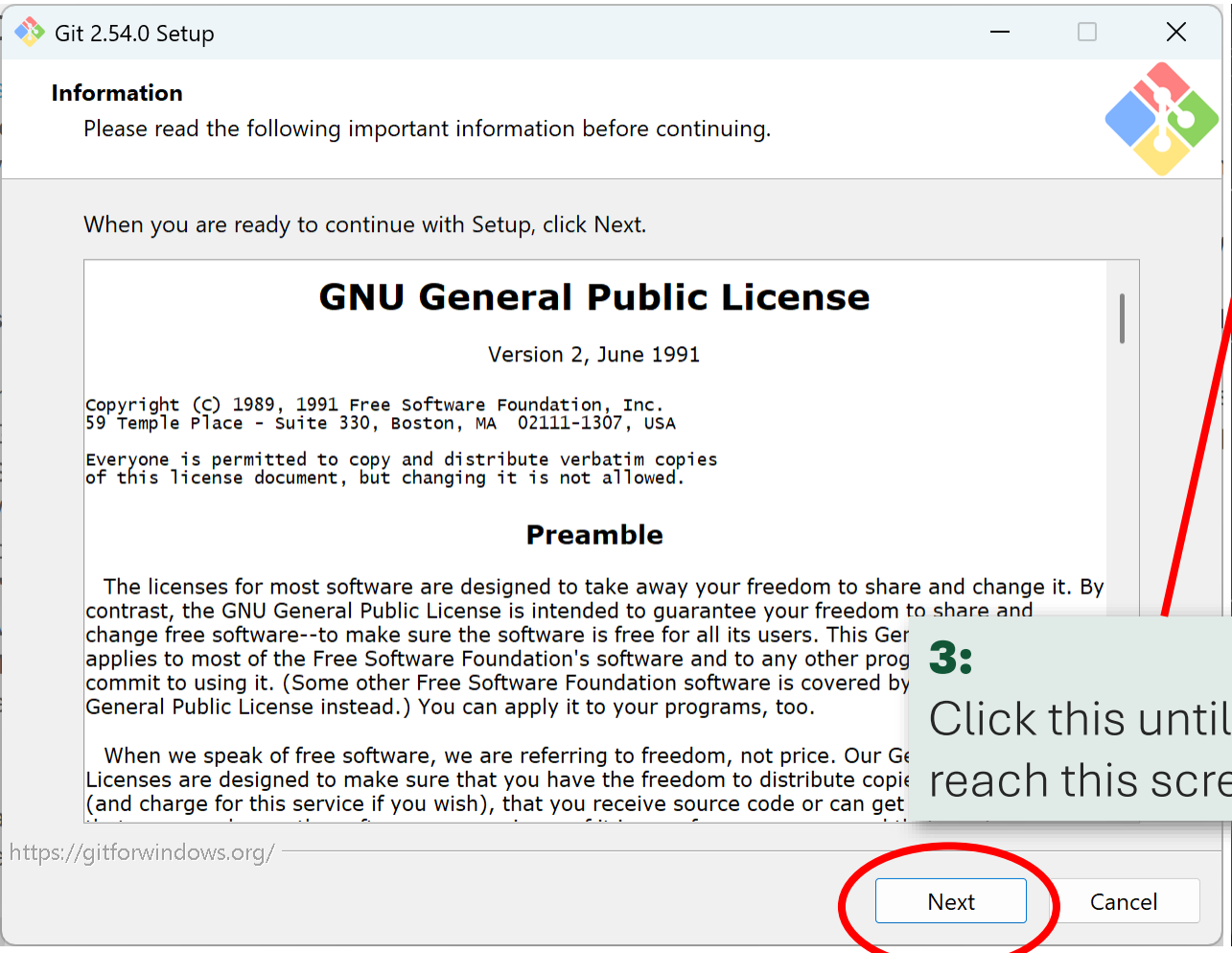
3:
If you do not have Homebrew installed, install it by navigating to “https://github.com/Homebrew/brew/releases/tag/5.1.13” and downloading the first link under “Assets”.
If you do have it installed, go to step **5**.

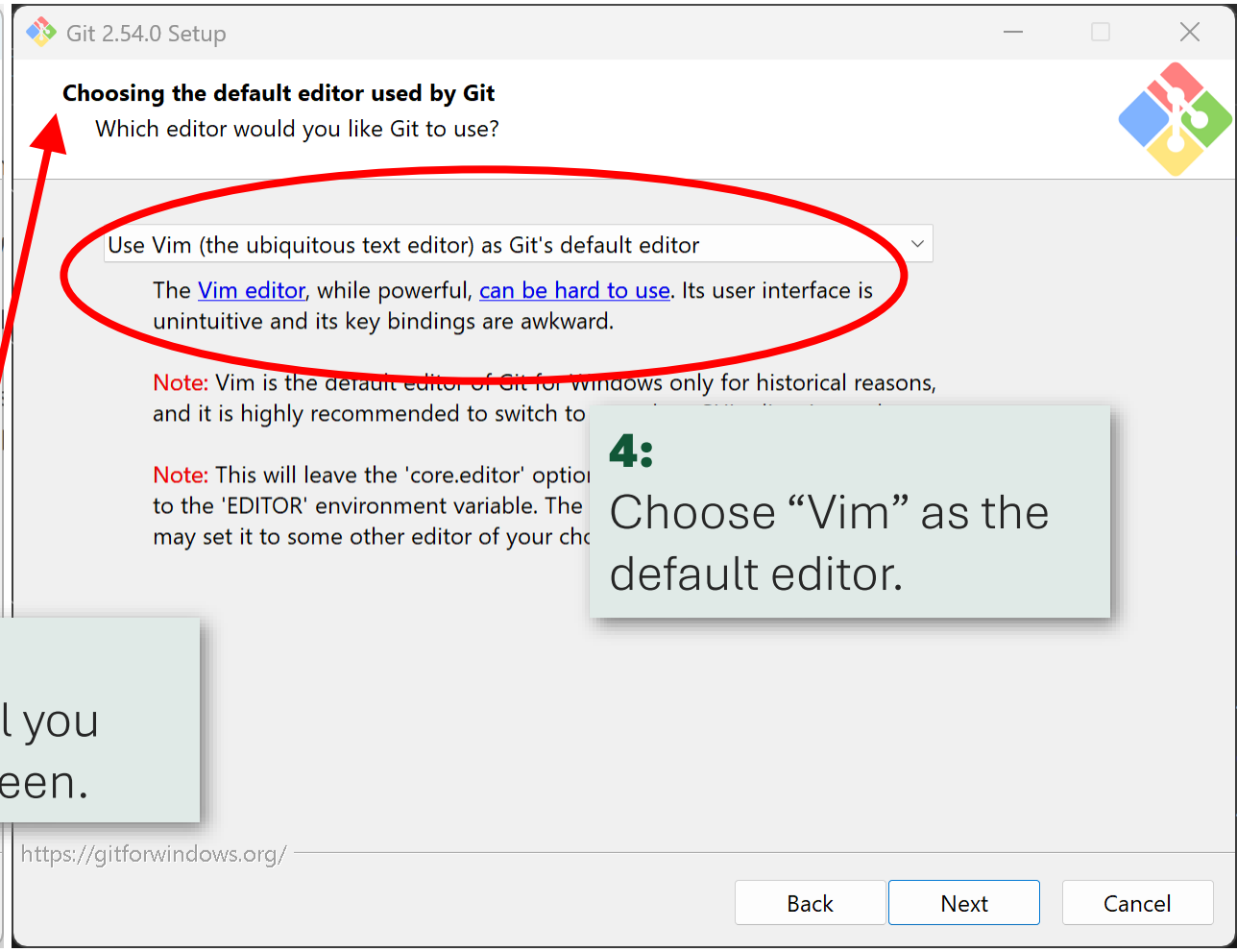
4:
Run “Homebrew.pkg” and follow instructions to download. (Ask Richard if you need assistance).

5:
Return to your terminal, and type `brew install git` to install Git.

Windows (Option 1): Install Git Bash

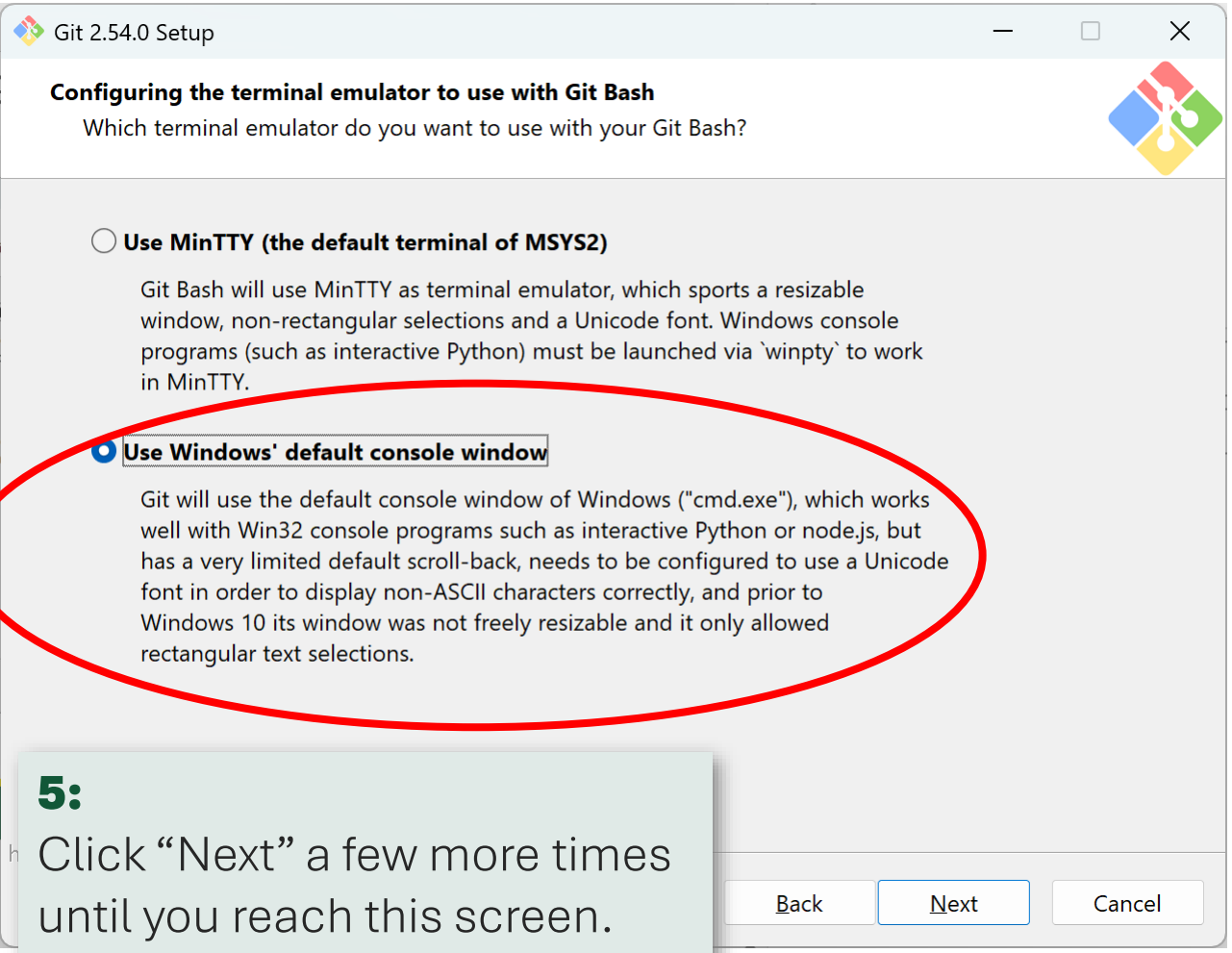




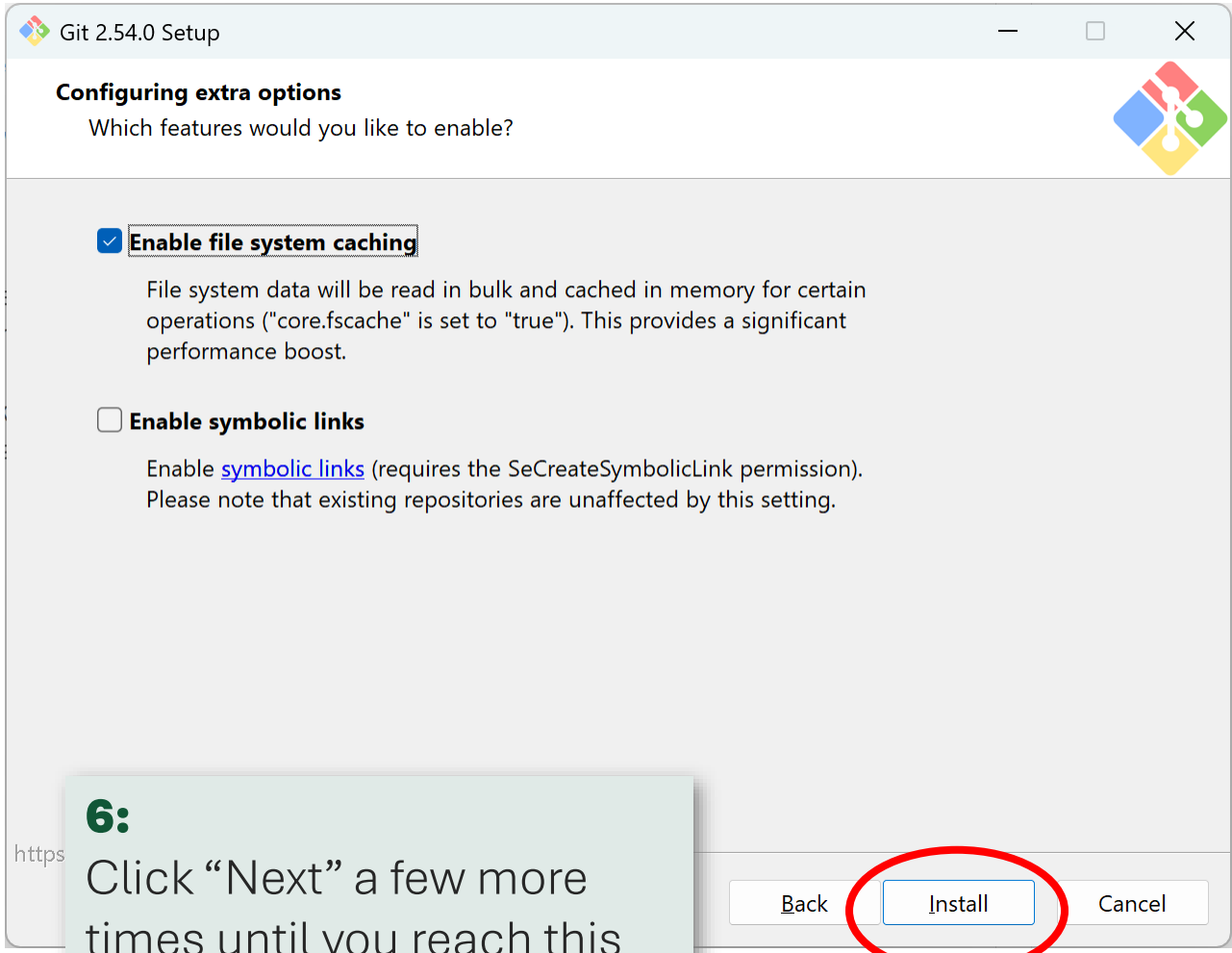


3:
Click this until you reach this screen.

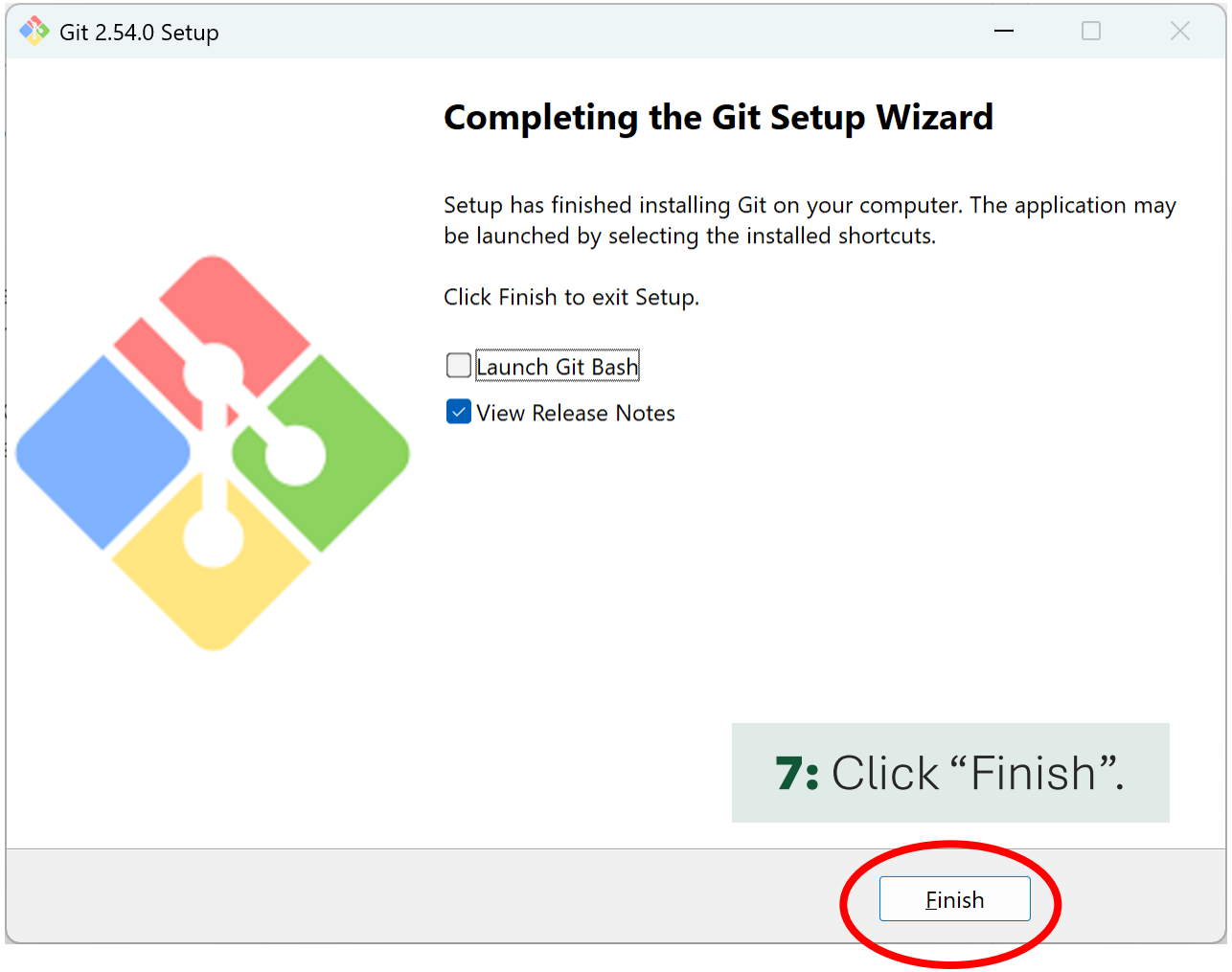
4:
Choose “Vim” as the default editor.

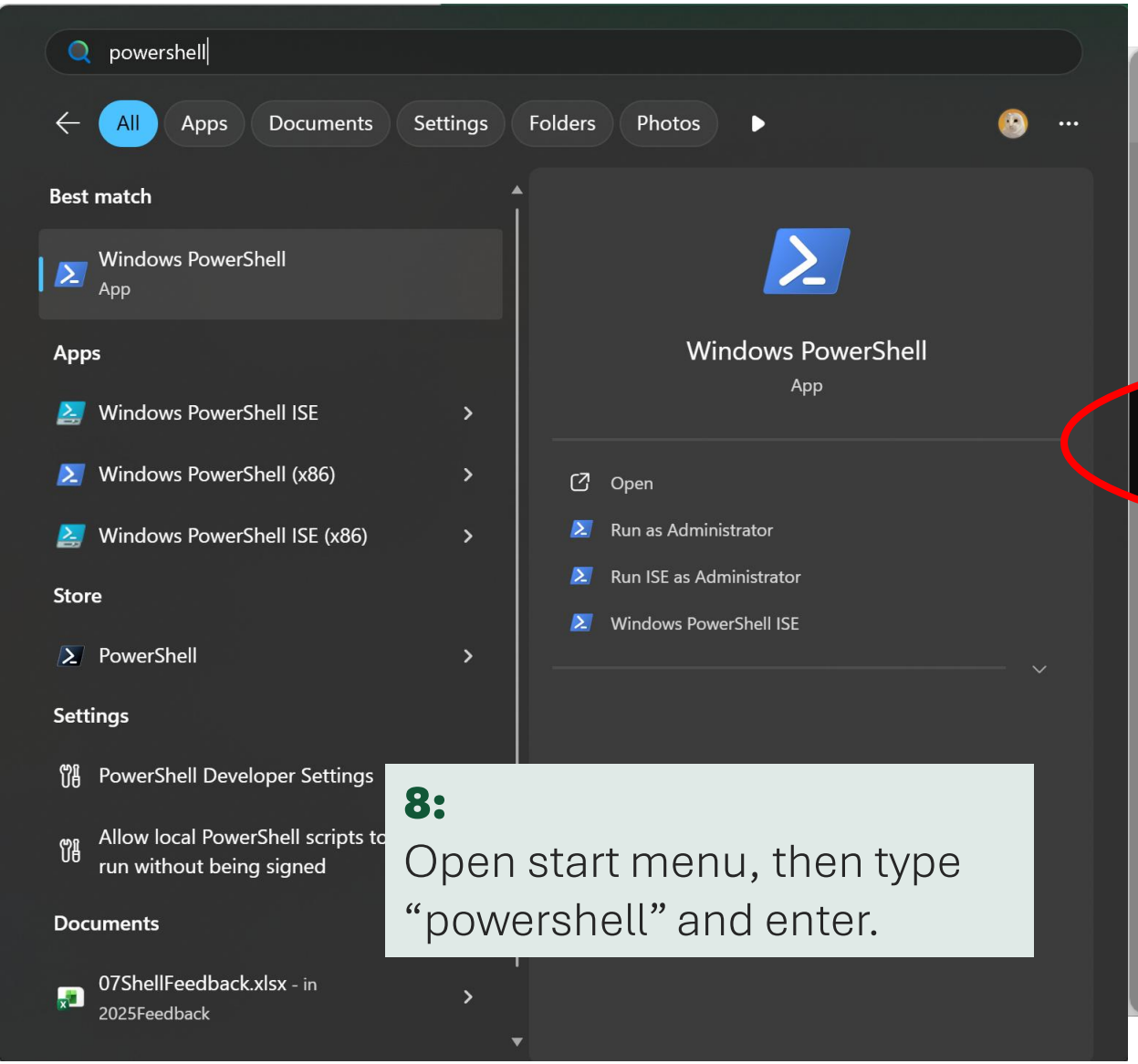


5:
Click “Next” a few more times until you reach this screen. Select second option.

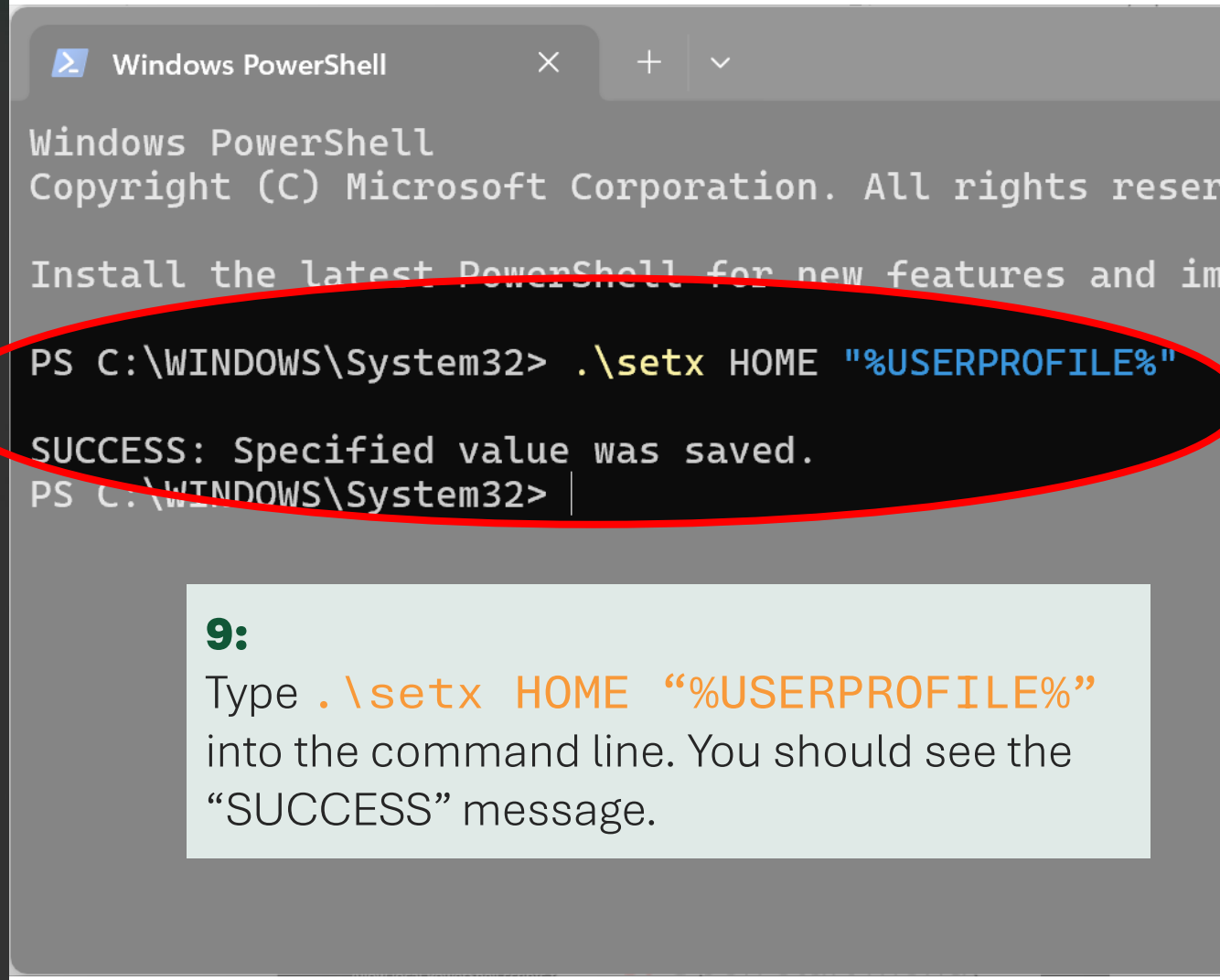


6:
Click “Next” a few more times until you reach this screen; then click “Install”.


Git 2.54.0 Setup
Completing the Git Setup Wizard
Setup has finished installing Git on your computer. The application may be launched by selecting the installed shortcuts.
Click Finish to exit Setup.
Launch Git Bash
View Release Notes
7: Click “Finish”.
Finish



8:
Open start menu, then type “powershell” and enter.

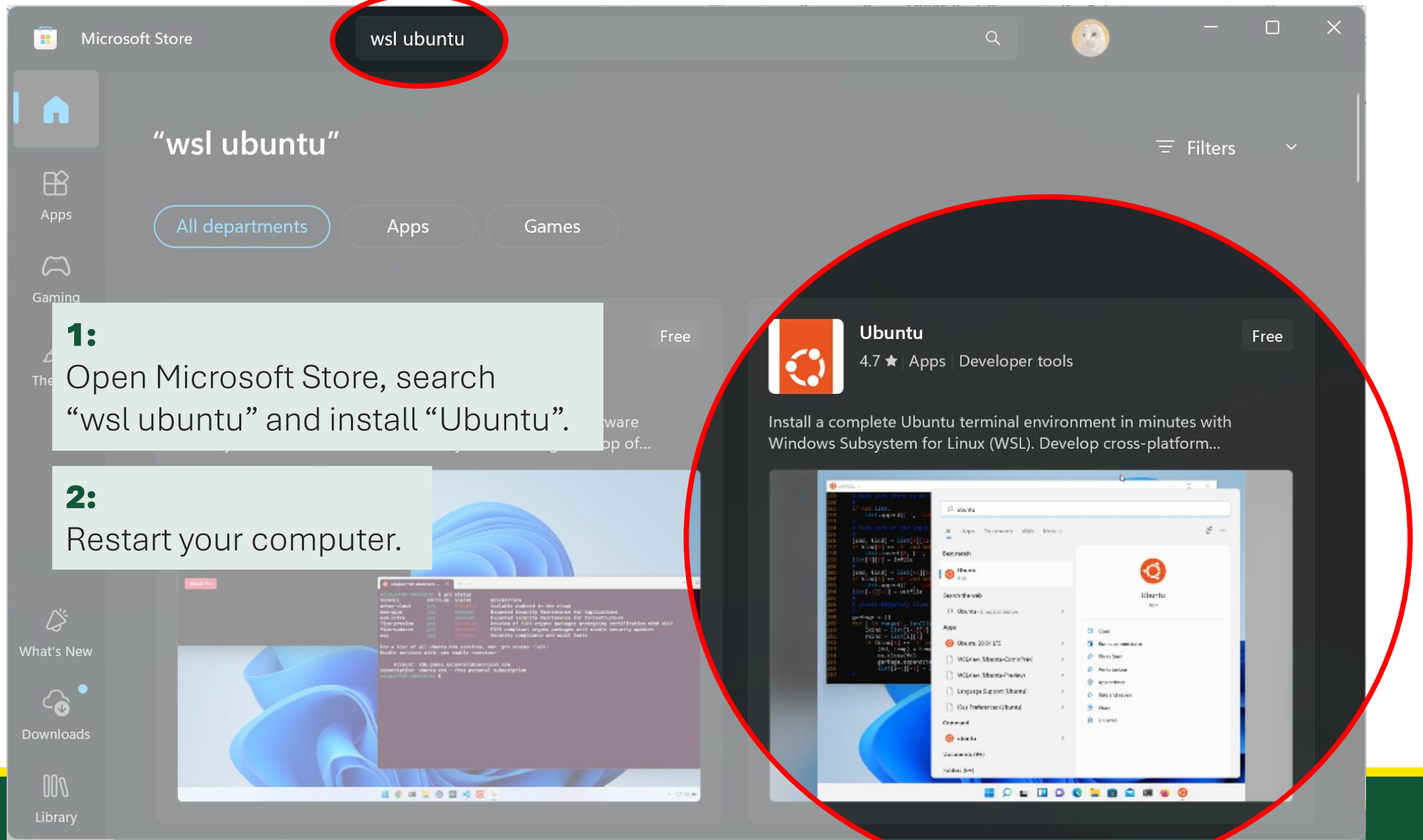


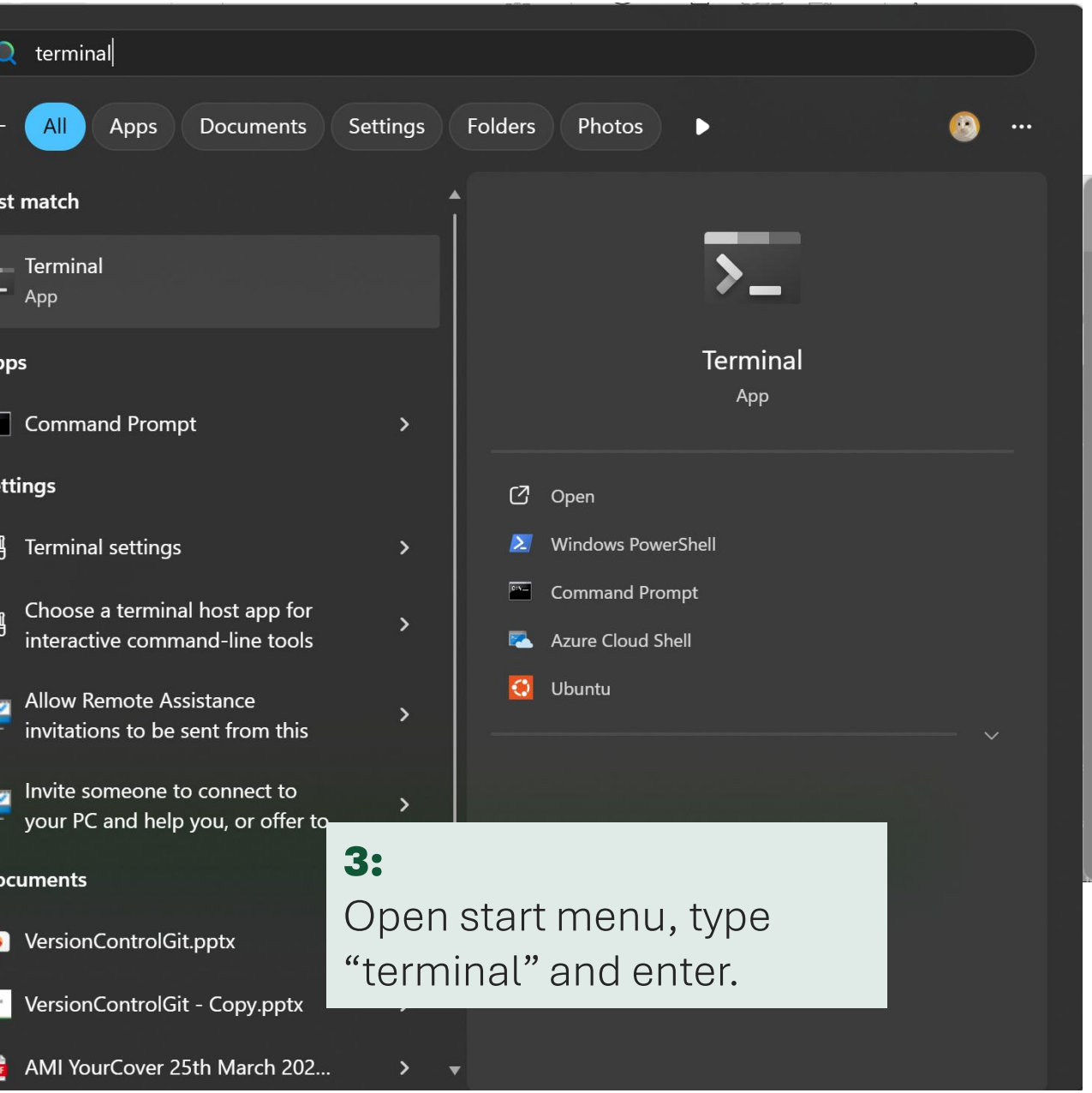
9:
Type `.\setx HOME “%USERPROFILE%”` into the command line. You should see the “SUCCESS” message.

Windows (Option 2): Install WSL

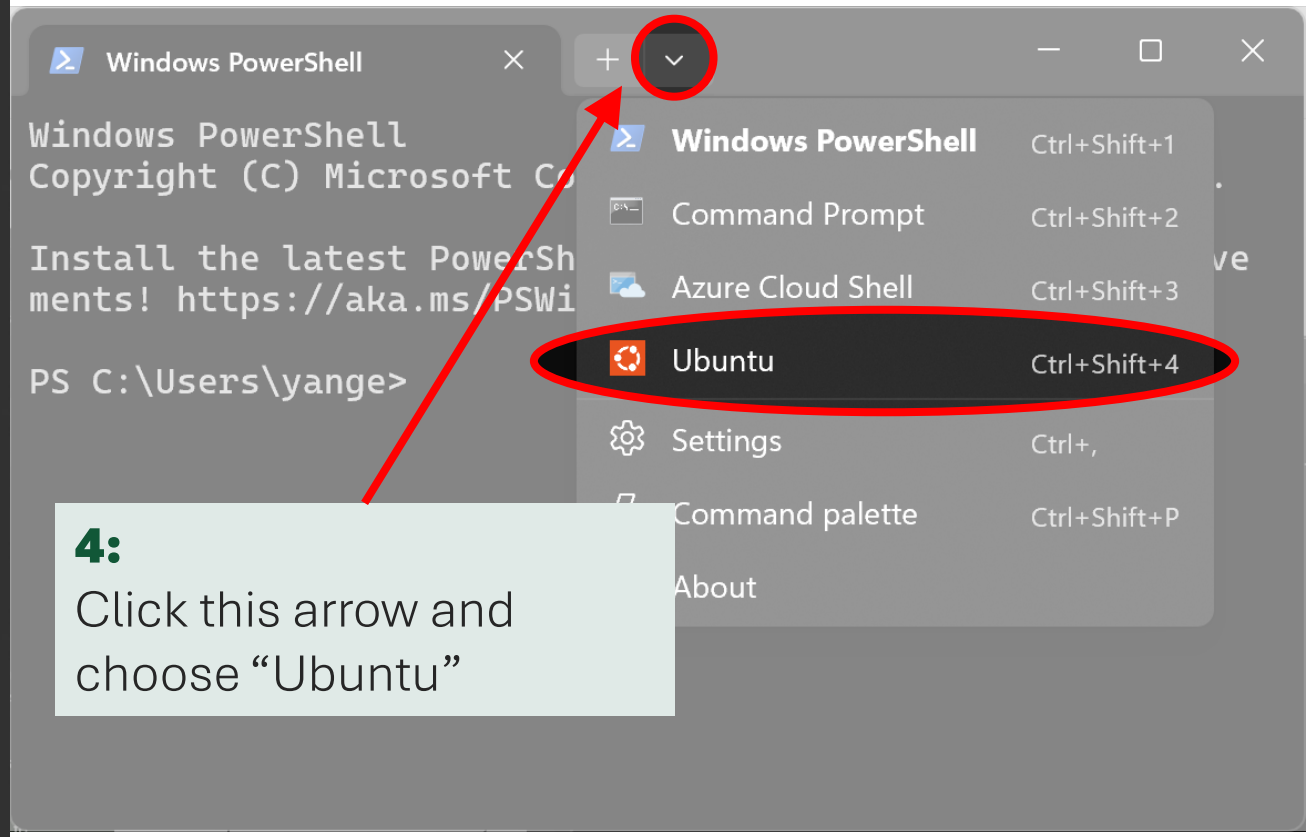
1:
Open Microsoft Store, search “wsl ubuntu” and install “Ubuntu”.

2:
Restart your computer.

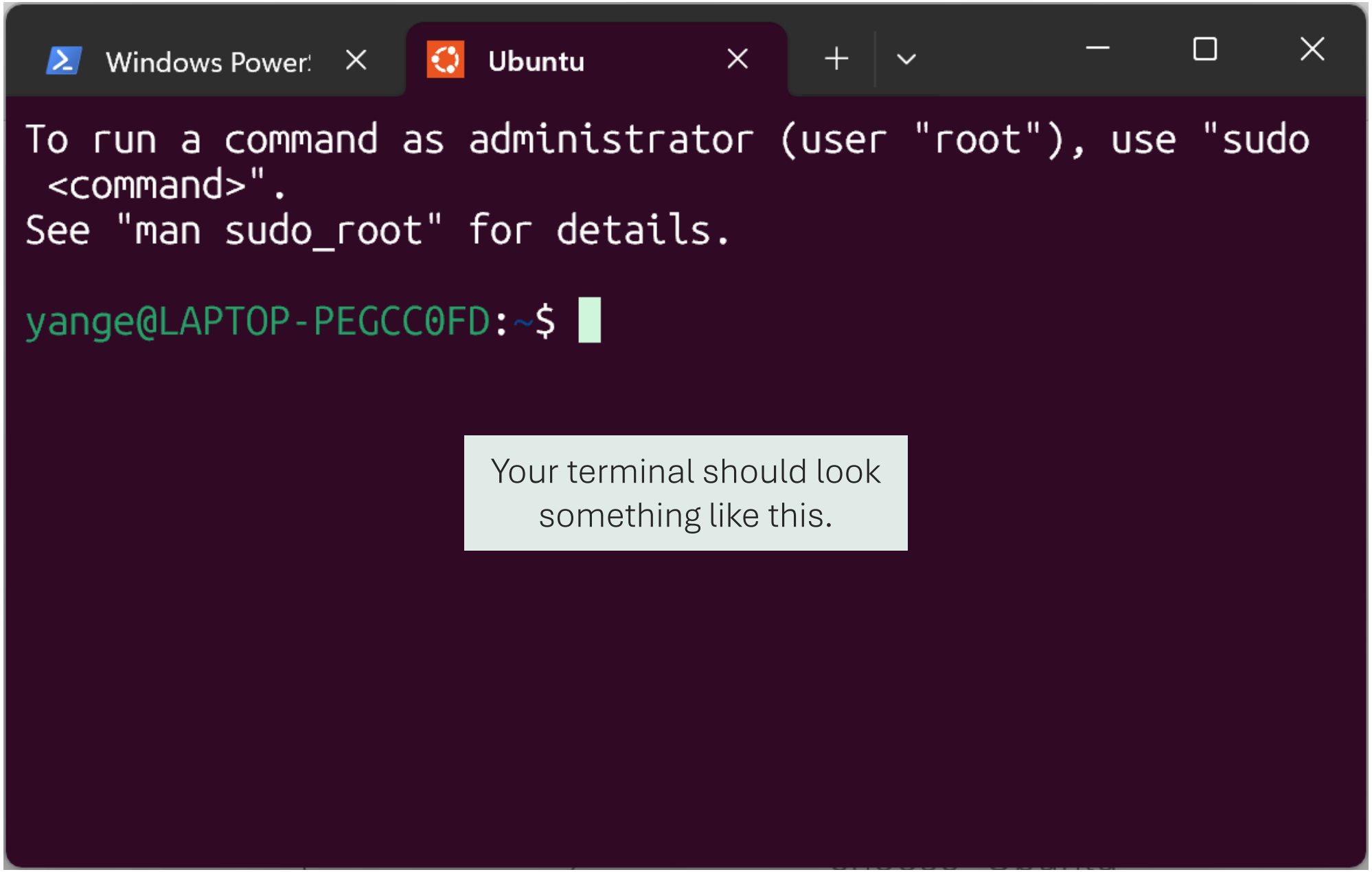


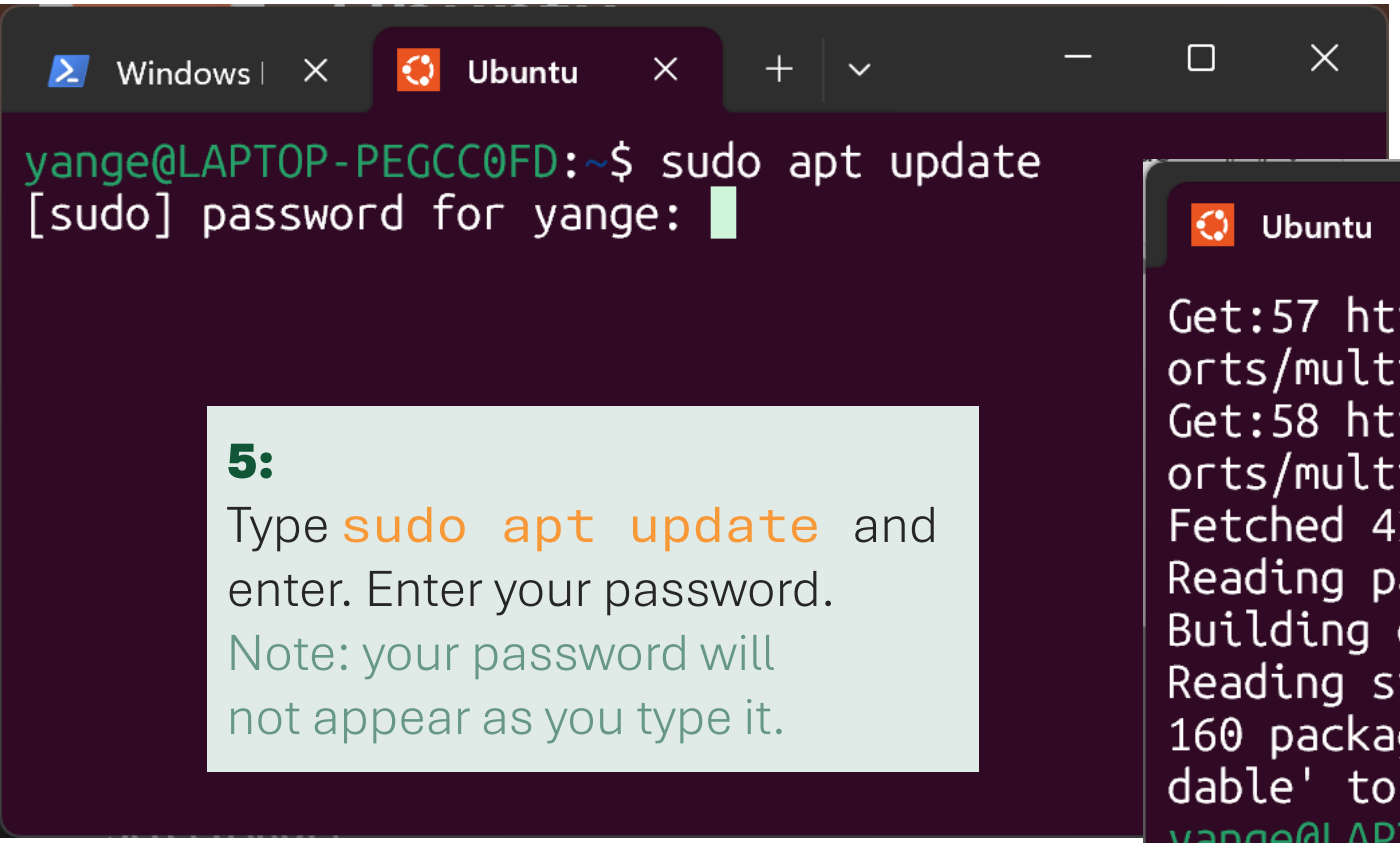


3:
Open start menu, type “terminal” and enter.

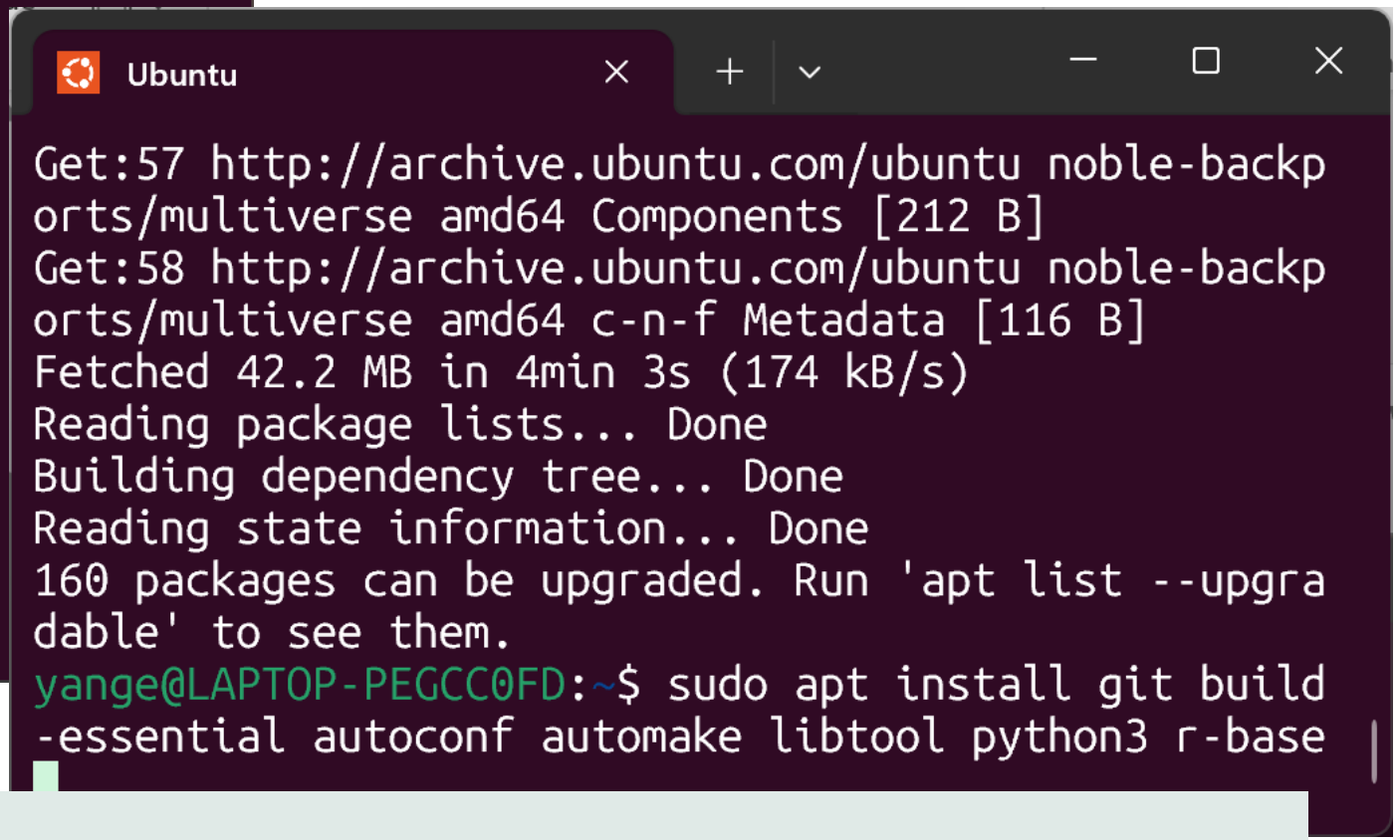


4:
Click this arrow and choose “Ubuntu”


Windows Power
Ubuntu
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
yange@LAPTOP-PEGCC0FD:~$
Your terminal should look something like this.



5:
Type `sudo apt update` and enter. Enter your password.
Note: your password will not appear as you type it.

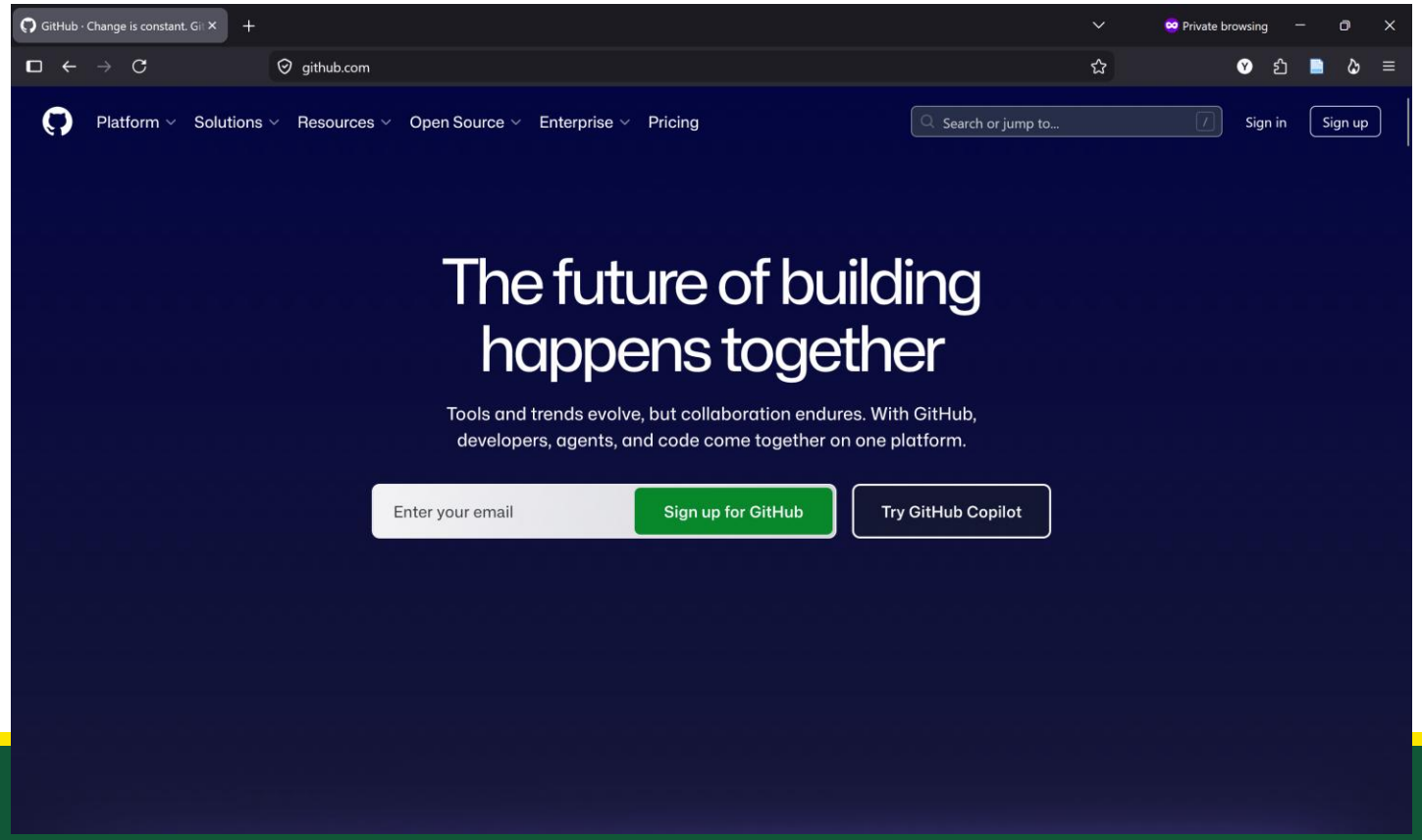


6:
Type `sudo apt install git build-essential autoconf automake libtool python3 r-base` and enter.
Type `y` and enter.

Create GitHub Account

Navigate to https://github.com/ and create an account if you don't already have one.

Note: GitHub required 2-factor authentication.



Open Terminal Application

For Linux: press “ctrl+alt+T” simultaneously.

For MacOS: search for “terminal” in the spotlight search bar and open the first option.

For Windows (Git Bash): in the start menu, search for “git bash” and open the first option.

For Windows (WSL): in the start menu, search for “powershell” and open the first option. On the top of the terminal window, click the down arrow and open an “Ubuntu” tab.

Part 1:
What and Why?

What is Version Control?

- A tool that records the changes and history of a specified project;
- Allows you to file old versions of a project and retrieve them as you please;
- Like saving a file, but on steroids.

Why use Version Control?

- An older version of a project has useful information that is not in your current working version;
- Your current version is broken, and you want to go back to an old, usable version;
- Unlimited “undo’s”;
- Avoid file names such as “thesis.doc”, “thesis_final.doc”, “thesis_FINAL.doc”, “thesis_FINAL_revised.doc”, “thesis_FINAL_revised_version6.doc”, etc…

You can also collaborate with others and have detailed change logs from contributors. We will not go over collaborations in this workshop.

Part 2: Setting Up Git

Configure Git

To save personalised settings

```
git config --global user.name “Your Name”
git config --global user.email “your.email@example.com”
git config --global init.defaultBranch main
git config --global core.editor “vim”
```

To see or edit all the globally configured values,

`git config --list --global` or

`git config --global --edit`

Historically, this has been called “master”, with connotations to slavery. Efforts are made to reduce this.

Choose your favourite editor. We will use Vim.

The flag “`--global`” tells Git to set these variables for all projects on your computer.

Git Help

Commands follow the form: `git <verb> <options>`

For help on a specific command:

```
git <verb> -h
git <verb> --help
```

Or more generally:

```
git help
```

Part 3:
Creating a Repository

Example: We are a Chef!

Navigate to your Desktop:	`cd ~/Desktop`
Make directory (folder) called ProjectReport:	`mkdir recipes`
Move into directory called ProjectReport:	`cd recipes`
Check which directory we are in:	`pwd`
See that it's empty:	`ls` or `ls -a`

Initiate a Git Repository

Initiate a Git repository

```
git init
```

Everything inside this folder will be tracked.

Show hidden .git folder

```
ls -a
```

This shows a hidden folder called ".git", which stores all the information involving this project and history.

Deleting ".git" will remove all version history information. The directory ceases to become a Git repository

Part 4: Tracking Changes

Check Status

```
git status
```

- This is a commonly used command to see latest changes.
- This will print an error if used outside of a Git repository.

Example: Guacamole Recipe

Create and edit a new file: `vim guacamole.md`

See that this new file is in the directory: `ls`

Check status of the repository: `git status`

```
$ git status
On branch main

No commits yet

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

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

Example: Guacamole Recipe

Tell Git to track this new file: `git add guacamole.md`

Show status: `git status`

Tell Git to commit this change to history: `git commit -m “Began outline of guacamole recipe”`

Show new status: `git status`

```
$ git status
On branch main
nothing to commit, working tree clean
```

Short description of changes you made. This will make it easier to search through past changes.

You can write longer messages with just `git commit`, which will take you to your default editor.

Show Past Changes

Show list of past changes in reverse chronological order:

```
git log
```

This will show the long identifier, where the first few characters correspond to the short identifier shown when you committed this change.

Also shown: author of the commit, time and date of commit, and commit message.

Example: More Changes

Add ingredients to your guacamole recipe: `vim guacamole.md`

Show status: `git status`

Show changes from last commit: `git diff`

The first few lines of this output are cryptic, referring to the file versions being compared. The last few lines will show the specific changes.

Example: More Changes

Remember to tell Git to add (stage) this new change:

```
git add guacamole.md
```

Commit the stage:

```
git commit -m “add ingredients for basic guacamole”
```

The `git add` feature is useful when we are changing more files. This way, we can choose which changes we want to commit at once, creating “snapshots” of our own choosing; e.g. changes that have a similar purpose or theme.

Example: Even More Changes

Change our recipe again:	`vim guacamole.md`
Stage this change:	`git add guacamole.md`
View differences:	`git diff`
View staged differences:	`git diff --staged`
Commit change:	`git commit -m “Make recipe more elaborate”`

Show Past Changes (again)

Show list of past commits: `git log`

Limit to the past N commits: `git log -N`

Limit to one line per commit: `git log --oneline`

When the log is too long, Git will page out the changes. To navigate the pager:

- "spacebar" to go to the next page,
- "Q" to quit the pager,
- To search for a specific word, type `/word` and "N" to flick through the matches.

A Note on Empty Directories

Git does not keep track of directories, only files within directories.

Git will not see when you create an empty directory, but will see when you create a new file in that directory.

To have Git track an empty directory, create a file called “.gitkeep” (or anything else) in that directory which Git will track.

Challenge 1

1. In your recipes folder, create 3 files, each containing an outline for a chocolate chip cookie recipe, lemon cake recipe or salad dressing recipe.
2. Stage all of these files.
3. Commit the stage with an appropriate message.

These changes are all similar, therefore it is appropriate to stage them together group them under the same commit and commit message.

Challenge 2: DIY

1. Create a new, empty directory, and create a Git repository out of it.
2. Create a new file in this directory. Add 3 fun facts about yourself. Commit this new file.
3. Of this file, modify this one line, add a new line.
4. Print differences in this file.
5. Commit the new change.
6. Show Git log.

Part 5: Exploring History

Example: Make a Bad Change

Add rat poison as an ingredient: `vim guacamole.md`

Do not stage or commit this change.

Compare to Previous Commits

Compare changes to last commit:

```
git diff guacamole.md
```
or
```
git diff HEAD guacamole.md
```

Compare changes to commit before last:

```
git diff HEAD~1 guacamole.md
```

Compare changes to 2 commits before last:

```
git diff HEAD~2 guacamole.md
```

Compare changes to Nth to last commit:

```
git diff HEAD~N guacamole.md
```

Compare changes to a specified commit:

```
git diff
ae7fec93cd004e886b3e0dceabeddb206b50333c
guacamole.md
```

This name can be got from the Git log.

Restore File to Previous Version

Restore file to previous commit:

```
git restore guacamole.md
```

Restore file to a specified commit:

```
git restore -s ae7fec93cd004e886b3e0dceabeddb206b50333c guacamole.md
```

Specific identifier

This changes a specific file to a previous version, from which you can stage and create a new commit from.

Example: Undo a Commit

You accidentally made a mistake in your files, staged and committed it. Therefore, you want to **revert** this commit. To do this, you do:

1. Look up identifier of previous commit: `git log -1`
2. Copy the identifier.
3. Revert to this commit: `git revert 7f742ed...8b6f6e`
4. Edit commit message.

This creates a new commit, which simply undoes the last. You can check this with `git log`.

Challenge 3

1. Create a new file called “ketchup.txt”.
2. In this file, write “I like tomatoes, therefore I like ketchup”
3. Type `git add ketchup.txt`
4. Edit “ketchup.txt” and replace the text with “ketchup enhances pasta dishes”.
5. Type `git commit -m “My opinions about the red sauce”`
6. Type `git restore “ketchup.txt”`

What text will “ketchup.txt” contain?

When Things get Complicated...

It is common for projects to be several months old and contain dozens of files. This makes it difficult to navigate through the project history.

Ways we can do this is by using `git log`.

Look up commits involving a specific file: `git log guacamole.md`

Look up commits and differences regarding a specific file: `git log --patch guacamole.md`

Part 6: Ignoring Things

.gitignore

Why don’t we want Git to track certain files?

- Files may be too big,
- Some files are temporary by nature,
- We can create some files using other files (e.g. a script can be used to download pictures),

Solution: we tell Git to ignore certain files.

Example: Ignore Pictures

Create a new directory called “pictures”: `mkdir pictures`

Populate “pictures” with random files:

```
touch pictures/cake1.jpg
pictures/cake2.jpg
```

Populate “recipes” with random files: `touch a.png b.png c.png`

Check status of Git repository: `git status`

```
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        a.png
        b.png
        c.png
        pictures/

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

Example: Option 1

Create .gitignore file:

```
vim .gitignore
```

Ignore all .dat files and .png files:

```
*.jpg
*.png
```

These lines are added into the .gitignore file.

Check status of Git repository:

```
git status
```

No comment about a.png, b.png, c.png, cake1.jpg, cake2.jpg are all shown.

Check status of ignored files:

```
git status --ignored
```

Force staging of an ignored file:

```
git add -f a.png
```

If you really want to track an ignored file, force this by running `git add -f a.png`

Example: Option 2

Create .gitignore file: `vim .gitignore`

Ignore everything in the data directory: `pictures or pictures/*`

Ignore all png files anywhere in repository: `**/*.png`

These lines are added into the .gitignore file.

Check status of Git repository: `git status`

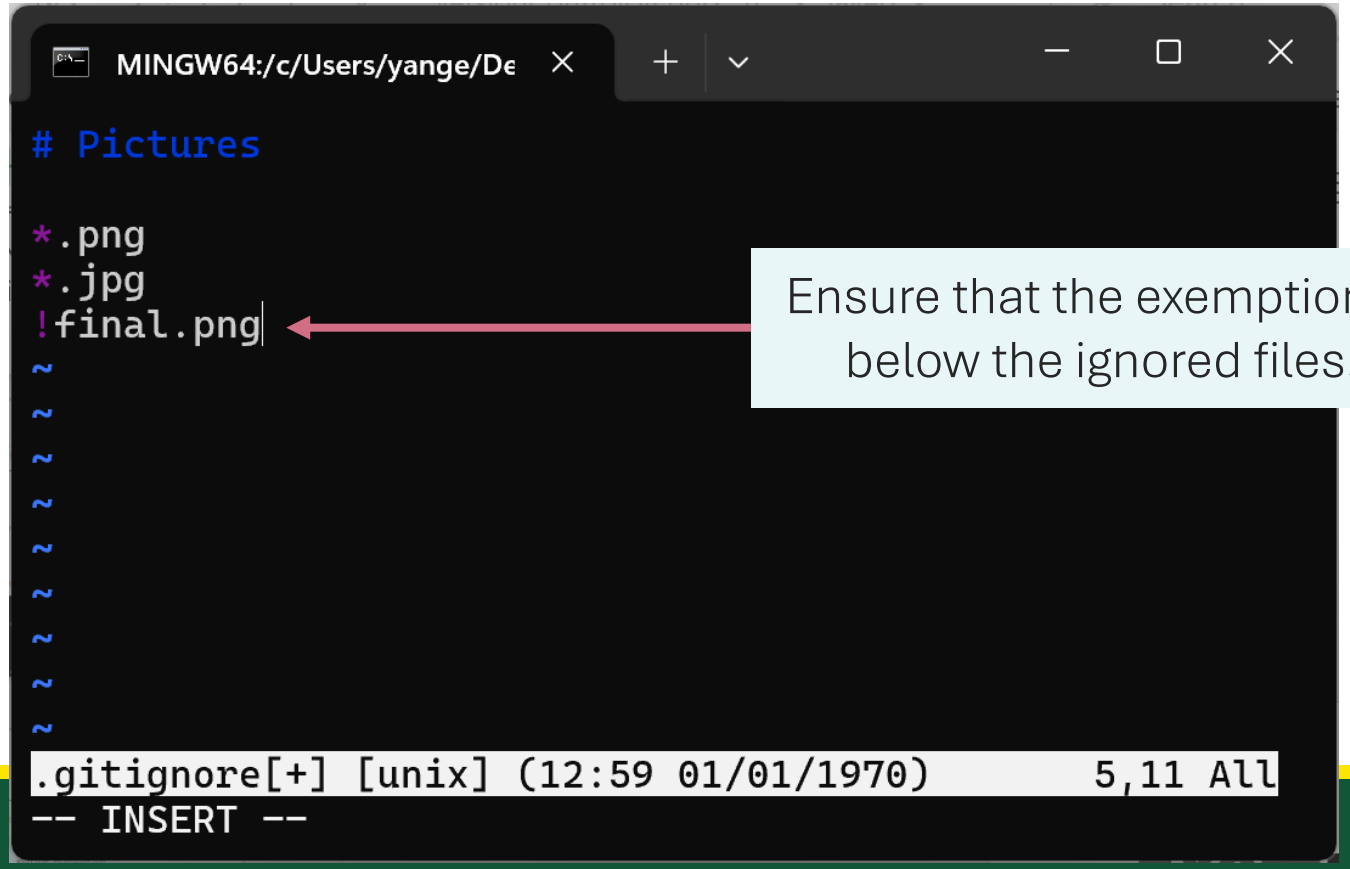
No comment about a.png, b.png, c.png, cake1.jpg, cake2.jpg are all shown.

Check status of ignored files: `git status --ignored`

If you really want to track an ignored file, force this by running `git add -f a.png`

Exemptions to Ignore

Let's say you want to ignore all images but 1 called "final.png". You may do this by adding an exemption in the .gitignore file with a "!" character.



Part 7: Remotes in Github

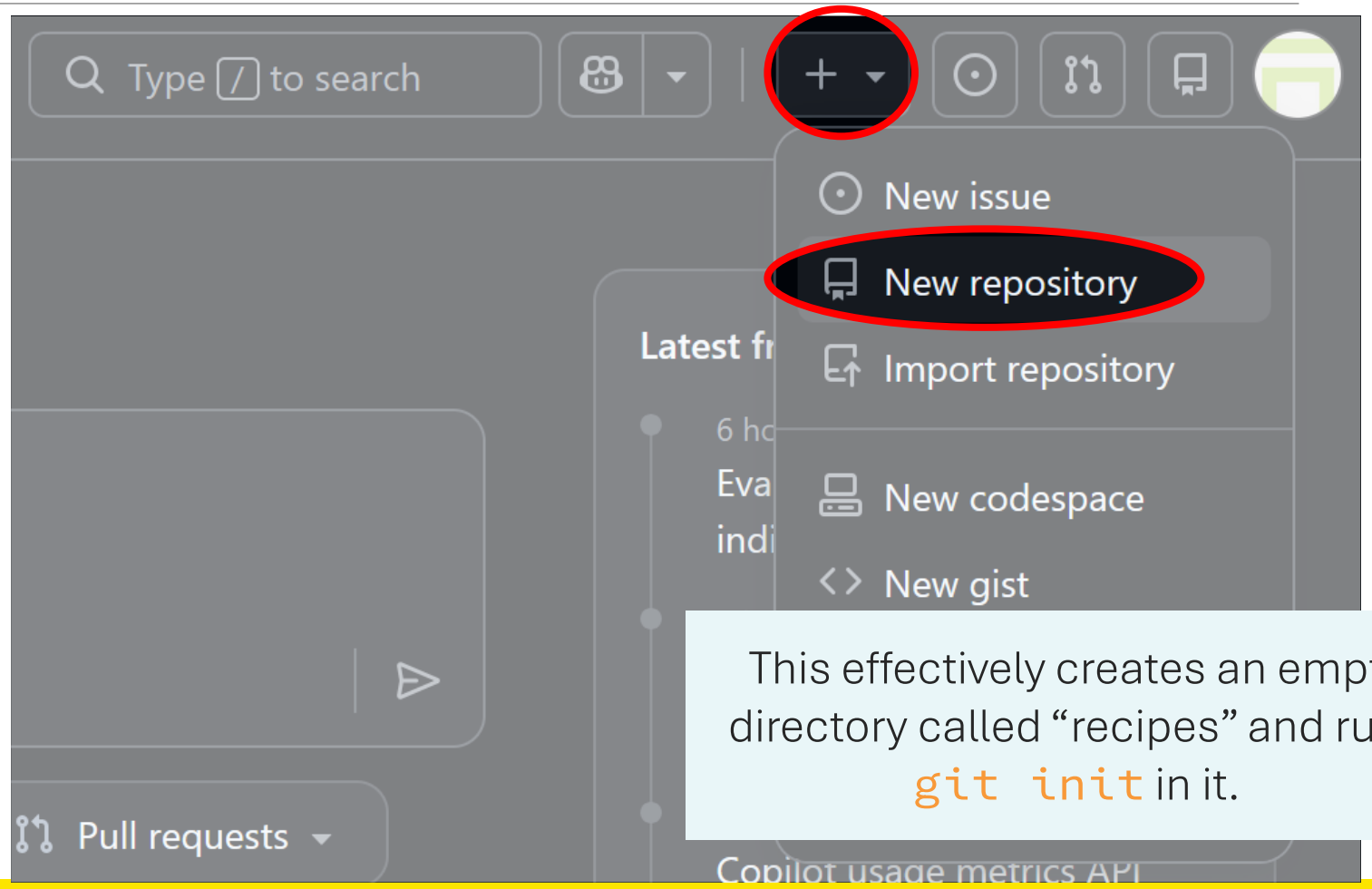
Create Remote Repository

1:
Go to “github.com”

2:
On the top right corner of the page, click the arrow next to the “+” button, and select “New repository”.

3:
Name your repository “recipes”.

4:
Leave all options unchanged. Click “Create repository” at the bottom of the page.

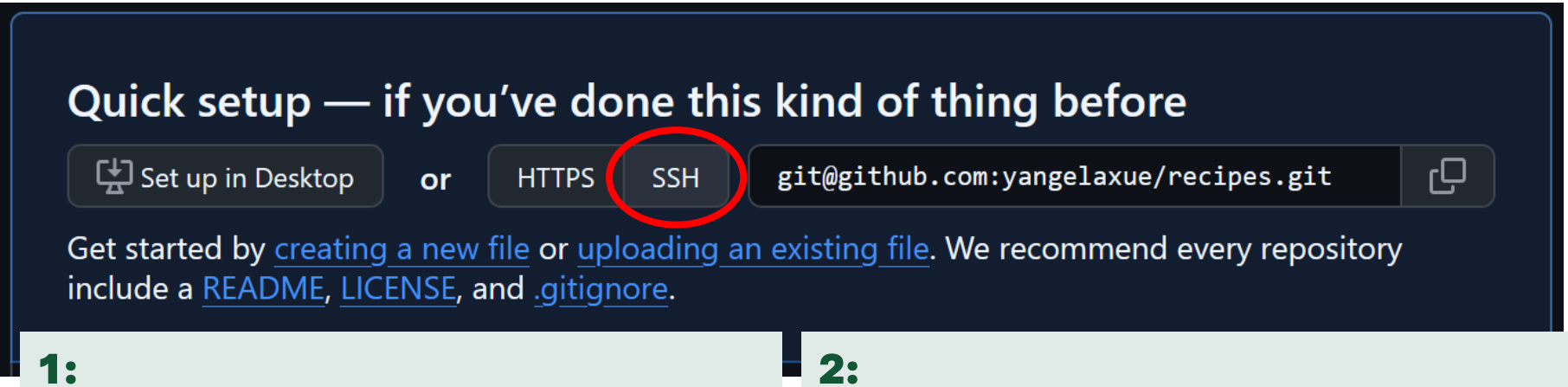


This effectively creates an empty directory called “recipes” and runs `git init` in it.

Connect Both Repositories

You now have 2 separate repositories with the same name.

The new GitHub repository will have instructions on how to connect the two.



1:
In your new, empty recipes repository on GitHub, click the SSH option. Copy this link.

2:
In your terminal, type

```
git remote add origin
git@github.com:yangelaxue/recipes.git
```

Setting up SSH

Before you can connect to your local repository, GitHub needs to know it is actually you who are connecting. We will use “Secure Shell Protocol” or SSH.

SSH works by having a public and a private key. GitHub can have a copy of the public key, and they will confirm that we can connect to it by having the private key stored on our device.

We will follow the minimum requirements needed to get this done.

1:
Type `ls -al ~/.ssh` into the terminal. If it lists files called “id_ed25519” and “id_ed25519.pub”, then you don’t need to do anything until step **6**. If not, move to step **2**.

Setting up SSH

Create a new key:

```
ssh-keygen -t ed25519
```

Or, create a new key and label it:

```
ssh-keygen -t ed25519 -C
“your.email@example.com”
```

2:
Create a new SSH key. The `-t ed25519` flag tells SSH to use the ed25519 algorithm.

3:
Enter to save key to the default location.

4:
Enter a password.
Note that it will not show as you type. If you are using a private computer, you can choose not to use a password.

Setting up SSH

```
Your identification has been saved in /c/Users/yange/.ssh/id_ed25519
Your public key has been saved in /c/Users/yange/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:hXUanVL/FkhfXvNfpsqHyCZSd3lKd6aw54YLoZNuEVA y.angela.xue@gmail.com
The key's randomart image is:
+--[ED25519 256]--+
|       .E oooo .o|
|      .  o.++.o.=|
|       .. o. ..o=|
|        ..    . ++|
|        S.o = + *|
|       ..= * O = |
|      . =.= *.+  |
|       o.+ ..+.  |
|       ..   .o.  |
+----[SHA256]-----+
```

You should get an output like this.

Setting up SSH

Now, we need to give GitHub our public key.

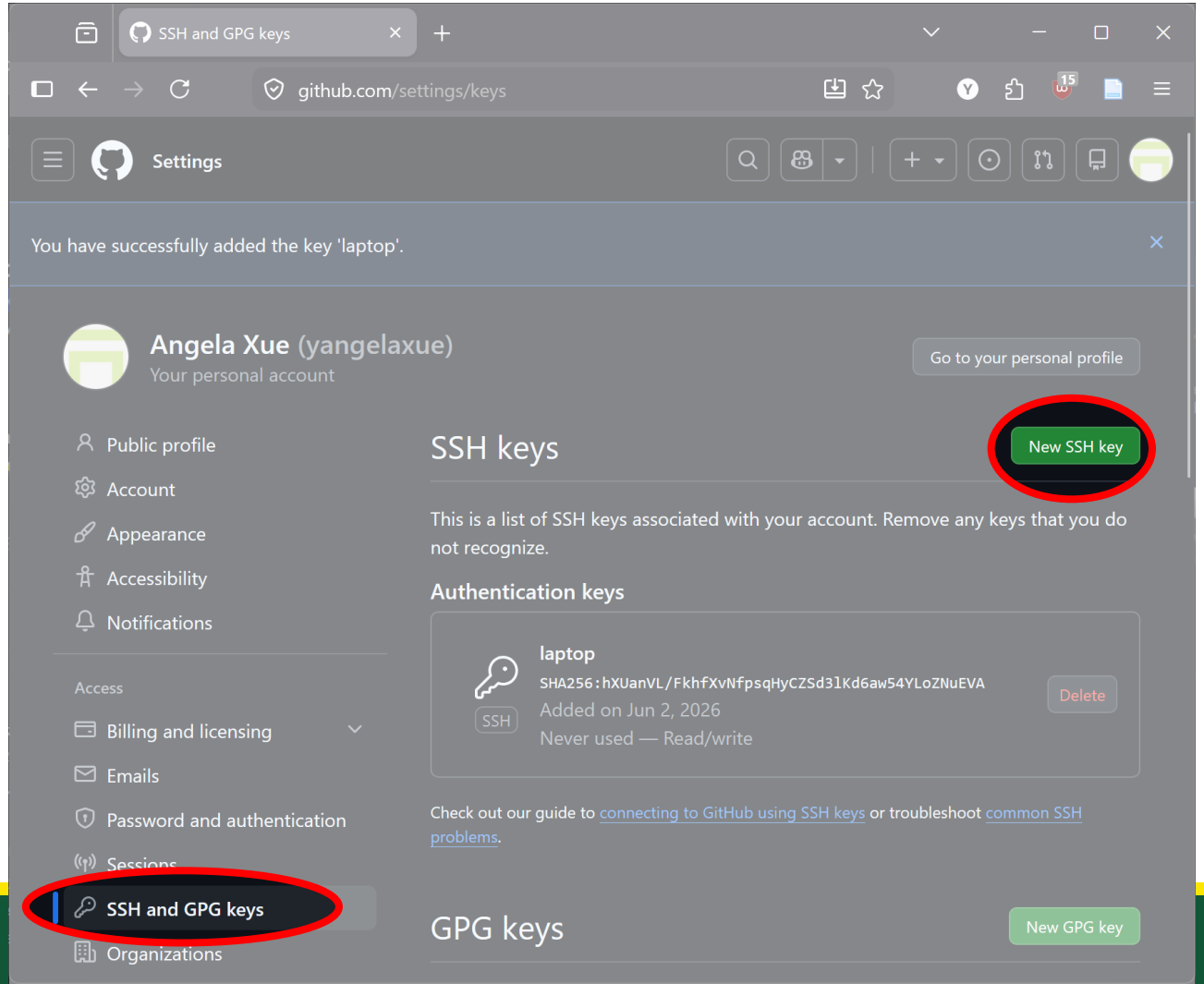
5:
Type `ls -al ~/.ssh` into the terminal. We should see a files called “id_ed25519” and “id_ed25519.pub”. The “.pub” extension means public.

```
$ ls -al ~/.ssh/
total 22
drwxr-xr-x 1 yange 197609    0 Jun  2 12:03 ./
drwxr-xr-x 1 yange 197609    0 Jun  2 11:28 ../
-rw-r--r-- 1 yange 197609  419 Jun  2 11:39 id_ed25519
-rw-r--r-- 1 yange 197609  104 Jun  2 11:39 id_ed25519.pub
-rw-r--r-- 1 yange 197609 1668 Jun  2 12:03 known_hosts
-rw-r--r-- 1 yange 197609  935 Jun  2 12:03 known_hosts.old
```

6:
Type `cat ~/.ssh/id_ed25519.pub` into the terminal and copy the output.

```
$ cat ~/.ssh/id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIPmpugI2vIUPDjBbjFbAC8+BlBnYFlrTQ0VFyJ3RztCd y.angela.xue@gmail.com
```

Setting up SSH



7:
Go to our GitHub settings, click “SSH and GPG keys” on the left hand panel, then click “New SSH key”.

8:
Title the key with a descriptive name and paste the public key in the “Key” section.

9:
Click “Add SSH key” at the bottom of the page.

10:
Enter `ssh -T git@github.com`
Into the terminal.

Pushing Changes to Remote

We can now push local changes to the remote branch, and we can pull changes from the remote branch to our local copy.

Push changes to remote: `git push origin main`

Pull changes from remote: `git pull origin main`

You can just run `git push` or `git pull` without any arguments by first running `git push –set-upstream-to origin main`.