GitHub x Python Workshop

Introduction to Git and GitHub

Hosted by IEEE CSUN
Student Branch

Contents

1   Introduction

Git is a distributed version control system that helps developers track changes in code, collaborate, and manage projects. This guide will cover all the basics of utilizing git. GitHub is a web platform that uses Git for version control and adds collaboration features. This guide will cover how GitHub is used to back up but also use it as a collaboration platform.

2   Setup

We need to first install a Git-bash:

- Windows: https://github.com/git-guides/install-git#install-git-on-windows

- Mac: Install GitHub from the App Store or using Homebrew: `brew install git`

- Linux:

    - If Debian-based: `sudo apt-get install git`

    - If Fedora-based: `sudo dnf install git`

    - If Arch-based: `sudo pacman -S git`

3   Verifying git Installation

Each Operating system has its own method to running its git client.

- Windows: Press the windows key and search up **Git Bash** and open the Git Bash application.

- Mac: Open up the terminal using: `CMD+Space` type in `Terminal`

- Linux: Open up the terminal using `Ctrl+Alt+T`.

In the terminal run `git --version` to verify the installation of git.

Figure 1: Printing git version

4 Basic Bash (Optional)

Bash is a programming language that can be used like python. Like how python commands can run on Python's interactive shell, Bash commands can also run in a shell. It runs in the (B)ourne (a)gain (sh)ell usually in your terminal. For many Linux distro's Bash is the default and for Mac its Zsh. For Windows it is Powershell. So in this workshop you will be learning a bit of Linux. Here is some list of basic commands:

- `cd`: short for change directory. Used to traversal through the directories in your system. Directories are just folders. Examples:

  - `cd ~` or just `cd` :This takes you back to your home directory. Each user on the PC has their own home directory.
  - `cd ~/Downloads/` :This will take you to your Downloads folder.
  - `cd ..` :This command take you back a single directory.
  - `cd ../..` :Takes you back two directories. You can add on to this to go back more directories.
  - `cd /` :Take you to the home directory of the system. Usually users can't write any files here.

- `ls`: Lists files and directories inside of your current directory. Examples:

  - `ls -a`: Lists all files and folders including hidden ones.
  - `ls ~/Downloads/`: It would print out everything in your downloads.
  - `ls -l ~/Downloads/`: It would print everything in a list format, all entries in a single format.

- `pwd`: short for print working directory. Typing just that print where you are right now.

- `cat`: concatenates/prints the file you gave it. Ex. `cat some_file.txt`: It would print out all that's in the file.

- `man`: Really useful to read manuals about a command. Ex. `man ls` tells you everything about the *ls* command.

- `ssh`: This command will allow us to remote access a server.

5 Configuring git

From now on all the steps for every operating system should be the same. Git requires its users to confirm people's identity to see who wrote the code. In the terminal type in

- `git config --global user.email "your_email_address_here"`

Following that set up your username:

- `git config --global user.name "your_user_name_here"`

Typing `git config --list` or `cat ~/.gitconfig` can help you verify if you set it up properly.

Note: Use your GitHub email and username to have a seamless transition when working with GitHub.

Figure 2: Configuring Git

6   Creating a Repository

After all this info dump of setting up git. Here is where we can get started. Log in to your GitHub account. Then, on the home page, click the **New** button. Give the repo the name **Python-RPS** and add a README file.
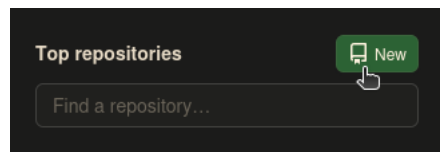

Figure 3: New repo button

Figure 4: Creating the Repo

7   Setting up Authentication

Now that you created the repository, you need a way to let GitHub know that the person accessing your repository is you. This is done by using private and public key pairs.

In the terminal enter the follow command:
`ssh-keygen -t ed25519 -C "your_github_email_here"`

There will be a prompt asking you to choose where to save the key pair, leave it at its default location by pressing **Enter.** A second prompt will ask to create a password to further encrypt the key pair, you can press enter to not add a password if you choose so.

Note: Having a password will ask you to enter the password every time you boot up your PC.

Figure 5: SSH key generating

For Windows Users: Enter following to start you ssh-agent: `eval "$(ssh-agent -s)"`
Now read the following steps like everyone else.

All Users: Add your private key to the ssh-agent using this command:
`ssh-add ~/.ssh/id_ed25519` Next step is to copy the public ssh key to GitHub. Firstly run `cat ~/.ssh/id_ed25519.pub` to print out the key and then head over to https://github.com/settings/keys or do it manually by clicking on the user profile then "Settings" then to "SSH and GPG keys"and add a new SSH key. To verify you have done all these steps correctly run `ssh -T git@github.com`
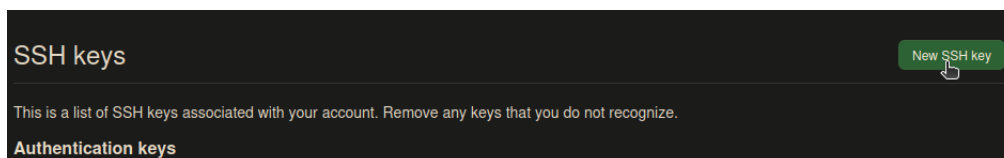

Figure 6: Creating a new key to GitHub

Figure 7: Adding Key to GitHub

Figure 8: SSH output when logging into GitHub.

8   Cloning a Repository

Head over to your GitHub Repository that you have created for this Python Project. There you will find a green button labeled "Code". Click this button and select the SSH key menu. Here you can copy the SSH key for your GitHub repo.

Syntax: `git clone URL_HERE`

Using the above you can clone the GitHub repo into your personal system. From here it is simply a matter of opening your programming IDE and creating a python file in the created directory.
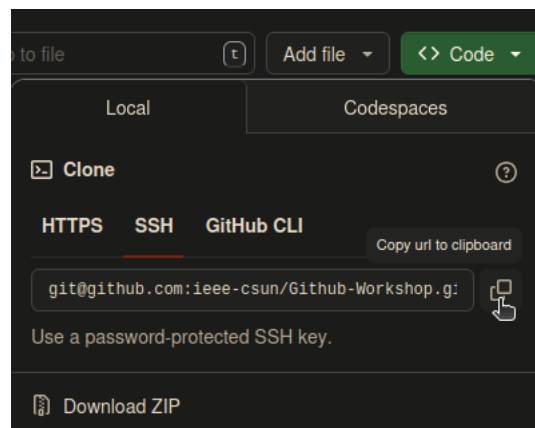
Figure 9: Copying the URL

Figure 10: Cloning a repo

9   Adding Local Repository (Optional)

Because of GitHub's capability of backing up your repo and encouraging collaboration between others. Commands to run:

- `git init` will change the directory you are on to a git repo. you can check by doing `ls -a` and look for a .git folder.

- Head over to GitHub to create a new **empty** repo. You will be syncing your local git repo to that remote repo.

- `git remote add origin <URL>` this will add the URL to your local repo. For us, use `git@github.com:USERNAME/REPO.git` because we setup ssh verification.

- Now make sure to `git add` your files and make a commit.

- Last step is to push the changes we do that by running. `git push -u origin main`

10   Storing Changes

When you have a git repo you are bound to make changes that you want to save and log so that you can revert changes later if something goes wrong. How do you save snapshots of your code? In git, we use `git commit`. To start off we run the command `git status`

-u to see if there are any changes that need to be saved. If there are unsaved changes we add the files to a staging area where the files are waiting to be commited. We do that by either using `git add <file_name>` or `git add -A` to commit all files. To commit our changes we do `git commit -m "a message about what you did"`. Final step is to store those changed made to the cloud, in our case to Github. We do that by running `git push` where we push the changes.

Note: Doing `git add -A` may add unwanted files. To avoid such scenario from ever happening we create a file called `.gitignore` where we can enter the file type, location, file name or all three combined should be ignored.

Figure 11: Git status simple with git add all

Figure 12: Git status detailed

Figure 13: Git commit example

Figure 14: Git push example

11　Frequently Used Commands

- `git pull`: Always to this when you start working of the project if you are collaborating .

- `git status`: This is to know what files are tracked and which files are not.

- `git add file_name` and `git rm file_name`: When you want to add or remove files from the project. `git rm` also deletes the file.

- `git commit -m "message"`: To backup changes to the code.

- `git push`: When you want to upload files to GitHub.

For more resources: https://education.github.com/git-cheat-sheet-education.pdf

12 Python Task

The following code is what you will be working to do today.

```python
import random

# Print game rules
print('Winning rules of the game ROCK PAPER SCISSORS are:\n'
    + "Rock vs Paper -> Paper wins \n"
    + "Rock vs Scissors -> Rock wins \n"
    + "Paper vs Scissors -> Scissors wins \n")

while True:

    print("Enter your choice \n 1 - Rock \n 2 - Paper \n 3 - Scissors \n")

    # TODO: Get and convert user input
    # HINT: Use input() to read from the user and wrap it with int()
    choice = ___

    # TODO: Keep asking until the input is valid (between 1 and 3)
    # HINT: Use a while loop to check if choice is not in the valid range
    while ___:
        choice = ___

    # TODO: Map the user's choice to its name
    # HINT: Use if-elif-else to assign choice_name based on choice number
    if choice == ___:
        choice_name = '___'   # HINT: This should be "Rock"
    elif ___:
        choice_name = '___'   # HINT: This should be "Paper"
    else:
        choice_name = '___'   # HINT: This should be "Scissors"

    print('User choice is:', choice_name)
    print("Now it's Computer's Turn...")

    # TODO: Computer makes a random choice
    # HINT: Use random.randint() with range 1 to 3
    comp_choice = ___

    # TODO: Map computer's choice to its name
    if ___:
        comp_choice_name = '___'
    elif ___:
        comp_choice_name = '___'
    else:
        comp_choice_name = '___'
```

```python
    print("Computer choice is:", comp_choice_name)
    print(choice_name, 'vs', comp_choice_name)

    # TODO: Determine the winner
    # HINT: First handle tie, then combinations where Paper, Rock, or
    ↪  Scissors win
    if ___:
        result = "DRAW"
    elif (choice == ___ and comp_choice == ___) or (choice == ___ and
    ↪  comp_choice == ___):
        result = '___'  # HINT: Paper wins in this case
    elif (___ and ___) or (___ and ___):
        result = '___'  # HINT: Rock wins in this case
    elif (___ and ___) or (___ and ___):
        result = '___'  # HINT: Scissors wins in this case

    # TODO: Print game result
    # HINT: Compare result with "DRAW" and with user choice_name
    if ___:
        print("<== It's a tie! ==>")
    elif ___:
        print("<== User wins! ==>")
    else:
        print("<== Computer wins! ==>")

    # TODO: Ask to play again
    # HINT: Use input() and convert to lowercase
    print("Do you want to play again? (Y/N)")
    ans = ___.___()
    if ans == 'n':
        break

print("Thanks for playing!")
```