



COLLEGE OF  
**ENGINEERING AND  
COMPUTER SCIENCE**

# Github x Python Workshop

Introduction to Git and Github

Hosted by IEEE CSUN  
Student Branch

# Contents

|   | Page     |
|---|----------|
| <b>1 Introduction . . . . .</b>               | <b>3</b> |
| <b>2 Setup . . . . .</b>                      | <b>3</b> |
| <b>3 Verifying git Installation . . . . .</b> | <b>3</b> |
| <b>4 Basic Bash (Optional) . . . . .</b>      | <b>4</b> |
| <b>5 Configuring git . . . . .</b>            | <b>4</b> |
| <b>6 Creating a Repository . . . . .</b>      | <b>5</b> |
| <b>7 Setting up Authentication . . . . .</b>  | <b>6</b> |
| <b>8 Cloning a Repository . . . . .</b>       | <b>8</b> |
| <b>9 Adding Local Repository . . . . .</b>    | <b>9</b> |
| <b>10 Storing Changes . . . . .</b>           | <b>9</b> |

# 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:

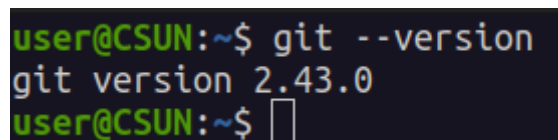
- Windows: <https://github.com/git-guides/install-git#install-git-on-windows>
- Mac: Install Github from the App Store or using Homebrew: `brew install git`
- Linux:
  - If Debian-based: `sudo apt-get install git`
  - If Fedora-based: `sudo dnf install git`
  - If Arch-based: `sudo pacman -S git`

# 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.



```
user@CSUN:~$ git --version
git version 2.43.0
user@CSUN:~$
```

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 it's Zsh. For Windows it's 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 travel 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 homes.
  - `cd ~/Downloads/`: This will take you to your Downloads folder.
  - `cd ..`: This command takes 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 prints 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.

```
user@CSUN:~$ git config --global user.email "ieee@my.csun.edu"
user@CSUN:~$ git config --global user.name "ieee_at_csun"
user@CSUN:~$ git config --list
user.email=ieee@my.csun.edu
user.name=ieee_at_csun
user@CSUN:~$
```

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.

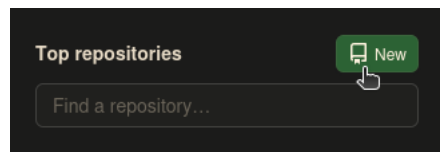


Figure 3: New repo button

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 keypair, 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.

```
user@CSUN:~$ ssh-keygen -t ed25519 -C "ieee@my.csun.edu"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/user/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_ed25519
Your public key has been saved in /home/user/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:eRAjRB7npu4nEHwSyTS/ONPBR5bba11cotmWeA+EzK0 ieee@my.csun.edu
The key's randomart image is:
+--[ED25519 256]--+
|  oo+*.Bo oo . |
|    += @ o+B.+ |
|   . * = +.O   |
|  +o.= =E+ o   |
|  +++ S o .    |
|  .+ . .       |
|  ..           |
|  ..           |
|  .o           |
+-----[SHA256]-----+
user@CSUN:~$
```

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`

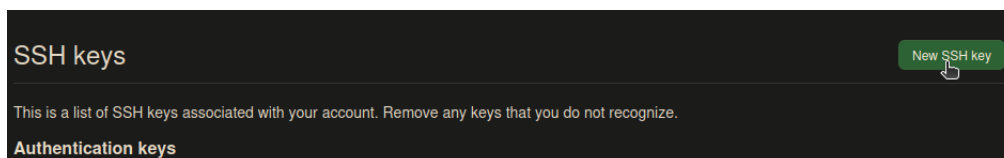


Figure 6: Creating a new key to Github

**Add new SSH Key**

**Title**  
My CSUN key

**Key type**  
Authentication Key

**Key**  
ssh-ed25519 AAAAC3CzaC1lZw11NTE5BBDk1JFC4WHQHGP9nWFB2/G02GUnPipma+aUPJmMmk/dvuk ieee@my.csun.edu

Add SSH key

Figure 7: Adding Key to Github

```
user@CSUN:~$ ssh -T git@github.com
Hi USER! You've successfully authenticated, but GitHub does not provide shell access.
user@CSUN:~$
```

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.



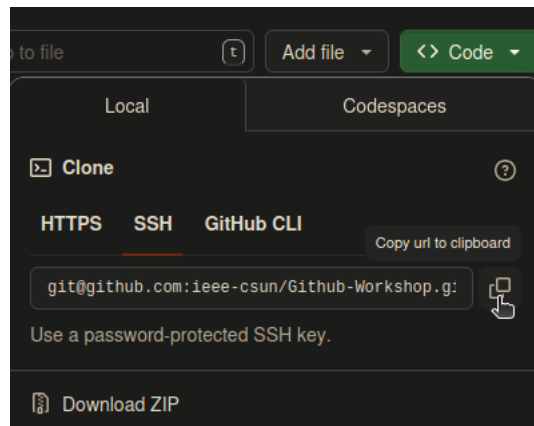


Figure 9: Copying the URL

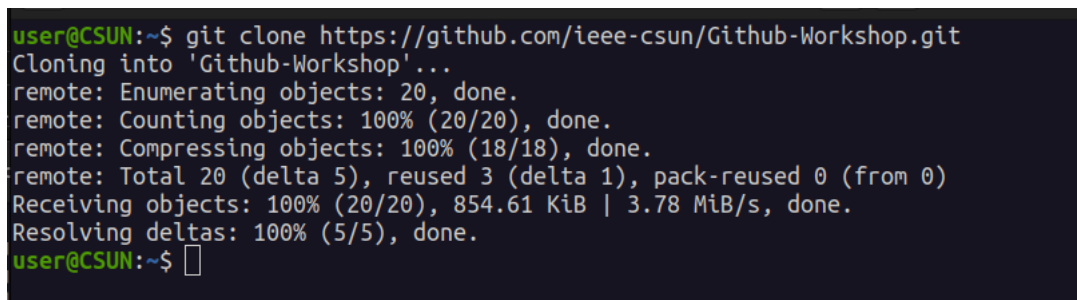


Figure 10: Cloning a reop

## 9 Adding Local Repository

Because of Github's capability of backing up your repo or 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 committed. 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 senario 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 ingored.

```
user@CSUN:~/Github-Workshop$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    GitHub_Workshop.tex

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    latex/GitHub_Workshop.tex

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        images/workshop-I/
        latex/GitHub_Workshop_I.tex

user@CSUN:~/Github-Workshop$ git add -A
user@CSUN:~/Github-Workshop$
```

Figure 11: Git status simple with git add all

```
user@CSUN:~/Github-Workshop$ git status -u
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    GitHub_Workshop.tex

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    latex/GitHub_Workshop.tex

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        images/workshop-I/GH-URL.png
        images/workshop-I/GH-add-key.png
        images/workshop-I/GH-keys.png
        images/workshop-I/GH-new-key.png
        images/workshop-I/GH-new-repo.png
        images/workshop-I/GH-setting.png
        images/workshop-I/git-clone.png
        images/workshop-I/git-config.png
        images/workshop-I/git-ver.png
        images/workshop-I/key-gen.png
        latex/GitHub_Workshop_I.tex
```

Figure 12: Git status detailed

```
user@CSUN:~/Github-Workshop$ git commit -m "feat: added in quality of life improvements"
[main 33c23fa] feat: added in quality of life improvements
15 files changed, 126 insertions(+), 232 deletions(-)
create mode 100644 .gitignore
create mode 100644 images/workshop-I/GH-URL.png
create mode 100644 images/workshop-I/GH-add-key.png
create mode 100644 images/workshop-I/GH-create-repo.png
create mode 100644 images/workshop-I/GH-new-key.png
create mode 100644 images/workshop-I/GH-new-repo.png
create mode 100644 images/workshop-I/git-clone.png
create mode 100644 images/workshop-I/git-config.png
create mode 100644 images/workshop-I/git-status-u.png
create mode 100644 images/workshop-I/git-status.png
create mode 100644 images/workshop-I/git-ver.png
create mode 100644 images/workshop-I/key-gen.png
create mode 100644 images/workshop-I/ssh-T.png
delete mode 100644 latex/GitHub_Workshop.tex
rename GitHub_Workshop.tex => latex/GitHub_Workshop_I.tex (55%)
user@CSUN:~/Github-Workshop$
```

Figure 13: Git commit example

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

```
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 = '___'

    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!")
```