

# Worksheet 01

Name: Mohit Sai Gutha UID: U48519832

## Topics

- Git

## Prerequisites (installations)

This is your checklist:

- [x] Access to terminal
- [x] Install Git
- [x] Sign up for a GitHub account
- [x] Choose editor
- [x] Set up ssh keys
- [x] Configure git

### Step 1: Work Environment: Access to Terminal

- Mac/Linux: use **Terminal**
- Windows:
  - Option 1: [Power Shell](#)
  - Option 2: Git Bash (recommended)

### Step 2: Install Git

- Mac:
  - [Git](#)
- Windows:
  - [Git for Windows \(Git Bash\)](#)
- Linux:
  - [Install Git on Linux](#)

Confirm Git is installed by typing `git --version` on your terminal

### Step 3: Sign up for a GitHub Account

Go to [github.com](https://github.com)

### Step 4: Choose a Graphical Editor

- Try Visual Studio Code
  - [Visual Studio Code](#)
- OR one of these other editors
  - [Sublime Text 3](#)
  - [Atom](#)
  - [Notepad++](#) (for Windows)

### Step 5: SSH Setup

#### Mac & Linux Users

Go to home directory (in terminal)

```
% cd ~
```

```
% pwd
/Users/gallettilance
```

Go to `.ssh` directory

```
% pwd
/Users/gallettilance
% cd .ssh
% pwd
/Users/gallettilance/.ssh
```

**Note:** If you do not have the `.ssh` directory, you can create it

- if you are in your home directory:

- `mkdir .ssh`

- if you are not in your home directory:

- `mkdir ~/.ssh`

Generate `id_rsa` keypair files if needed

- **Note:** these `id_rsa` files contain a special password for your computer to be connect to network services (Ex: GitHub, AWS).

- Check to see if these files exist by typing `ls -alt`

- If you do not have these two files (`id_rsa` and `id_rsa.pub`), create them by typing:

- `ssh-keygen`

- Hit `enter` 3 times

```
% pwd
/Users/gallettilance/.ssh
% ls
% ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/gallettilance/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/gallettilance/.ssh/id_rsa.
Your public key has been saved in /Users/gallettilance/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:jmDJes1qOzDi8KynXLGQ098JMSRnbIyt0w7vSgEsr2E gallettilance@RESHAMAs-MacBook-P
ro.local
The key's randomart image is:
+---[RSA 2048]-----+
|    .+=          |
| .   .==         |
| .o  +o          |
|..+= oo          |
|.E.+X.  S        |
|+o=o=*oo.        |
|++.*o.+o.        |
|..*.oo           |
|o= o+o           |
+----[SHA256]-----+
% ls
total 16
-rw-----  1    1675 Dec 17 12:20 id_rsa
-rw-r--r--  1    422 Dec 17 12:20 id_rsa.pub
%
```

Navigate to the `.ssh` directory

```
cd ~/.ssh
```

open `id_rsa.pub` using your editor of choice and copy its contents. Add `ssh` key to GitHub by following these steps:

- go to your [GitHub account](#) (create one if you don't have one, and save your user name and password somewhere easily accessible for you.)
- click on your avatar/profile picture (upper right of screen)
- go to `Settings`
- on left of screen, select `SSH and GPG keys`
- Select `New SSH key`
- for "Title": entitle it "GitHub key"
- for "Key": paste key from clipboard here
- click `Add SSH key`
- save, exit, confirm GitHub password as requested

## Windows Users

Follow [How to Create SSH Keys with PuTTY on Windows](#)

### For Windows 10 or 11

- Press Windows+R
- Enter cmd
- In the opened Command Prompt, type in “ssh-keygen”
- Press Enter
- You can choose to enter a passphrase, it will not be displayed.
- Go to the shown path to find your file named `id_rsa.pub` Ex. `C:\Users\user\.ssh\id_rsa.pub`
- Open the file with a notepad and copy everything
- Go to github and click settings at top right
- Go to SSH and GPG keys, click New SSH key and paste your SSH key here.
- Click Add SSH key. You might be asked to enter Github password.
- Go back to your Command Prompt and type in “ssh -T git@github.com”
- Enter your SSH passphrase.
- You will see “You’ve successfully authenticated” in the following message. Which means you are now connected to Github.

## Step 6: Configure Git

### Configure user name and email (lets Git know who you are)

```
git config --global user.name "First Last"
```

```
git config --global user.email "myname@email.com"
```

To verify these additions, type:

```
git config --list
```

### Default Editor

The default editor will be [Vim](#). You may want to look up how to edit, save, and close vim as this can't be done with just point and click (you must use the vim commands).

## Git / GitHub (In Class)

a) what is the difference between git and github?

Git is a version control system that can be used to keep track of changes made to files (typical used by developers to track their code). Git hub as the name suggest is a hub or a platform where we can host these git files called git repositories. Users on github can collaborate and work on git repositories while maintaining the entire timeline of the repositories.

b) what command would you use to conv a repo locally?

git clone 'url of the git repository'

c) what button would you use to make a copy of a repo in GitHub?

'Fork' button on the repository page on github.

d) let's say you have a copy of a repo in GitHub but that repo changes, does your copy on your laptop change too? why / why not?

The changes are never affected on your local copy automatically. We need to 'pull' the changes to update the copy on our laptop using the command 'git pull'.

e) what are the three commands you use to create a new save point in your git repo and back it up to GitHub?

add - a change in the working area is added to the staging area commit - commits the change (saves the change basically) push - pushes/uploads the changes to the github repo

f) how would you make your local and remote copies change so that they have the most up-to-date version of the repo they are copied from?

We would use git pull to pull the changes from the repo to update and keep the local and remote copies up to date.

g) why are there sometimes conflicts between copied repos / branches? How do you resolve them?

When changes in one copy cannot be directly merged with changes in another copy due to the change being either made in the same region or functionality that is affected by it. In such cases we have to manually edit the files to best suit our needs and decide on what to keep and what to discard to resolve these conflicts.

h) describe all the steps needed to make a PR to contribute your notes to the class repository.

1. Fork the class repository to personal github account.
2. Clone forked repository to local machine.
3. Create new branch to contribute to class notes.
4. After having created necessary files and content, add, commit and push changes to newly created branch of the forked repository.
5. Go to the original repo, in pull requests tab, choose new pull request button and choose the repo and branch to affect change from.

i) Write here some other commands we used in class and what they mean / how to use them:

1. git diff - It shows the difference between the working area, staging area and the last commit. for example to see the changes between (difference between) two specific commits, commit1 and commit2, we can use the command - git diff commit1 commit2
2. git rebase - It is used to rewrite commit history by moving or combining changes. for example to rebase a current branch onto another branch we use the command - git rebase

## Exercise

a) Create a public repo on your github called "polynomial". Create a folder on your laptop called "polynomial" and initialize it as a git repo. Add a remote called "origin" pointing to the github repo you just created. Create a file called "polynomial.py" with the following code:

```
class X:
```

```

def __init__(self):
    pass

def __repr__(self):
    return "X"

class Int:
    def __init__(self, i):
        self.i = i

    def __repr__(self):
        return str(self.i)

class Add:
    def __init__(self, p1, p2):
        self.p1 = p1
        self.p2 = p2

    def __repr__(self):
        return repr(self.p1) + " + " + repr(self.p2)

class Mul:
    def __init__(self, p1, p2):
        self.p1 = p1
        self.p2 = p2

    def __repr__(self):
        if isinstance(self.p1, Add):
            if isinstance(self.p2, Add):
                return "( " + repr(self.p1) + " ) * ( " + repr(self.p2) + " )"
            return "( " + repr(self.p1) + " ) * " + repr(self.p2)
        if isinstance(self.p2, Add):
            return repr(self.p1) + " * ( " + repr(self.p2) + " )"
        return repr(self.p1) + " * " + repr(self.p2)

poly = Add( Add( Int(4), Int(3)), Add( X(), Mul( Int(1), Add( Mul(X(), X()), Int(1)
))))
print(poly)

```

add and commit this file with the following message "cs 506 exercise part a". Push these changes to github.

b) In this exercise, you will write code to define and evaluate polynomial expressions. You should write out polynomials yourself to test various use cases / edge cases. Using the code above as an example, write classes for:

- division (called `Div`)
- subtraction (called `Sub`)

you may modify the `Mul` class if needed. Ensure that the rules of parentheses are properly encoded in your `repr` functions. When you're done with this part, add and commit the changes with the message "cs 506 exercise part b". If you need to modify the code from this exercise at a later time, you must use the interactive rebase so that all changes related to this section are in the relevant commit.

c) Write an `evaluate` method to each class that can evaluate the polynomial for a given value of `X`.

```

poly = Add( Add( Int(4), Int(3)), Add( X(), Mul( Int(1), Add( Mul(X(), X()), Int(1)
))))
print(poly.evaluate(-1))

```

When you're done with this part, add and commit the changes with the message "cs 506 exercise part c". If you need to modify the code from this exercise at a later time, you must use the interactive rebase so that all

changes related to this section are in the relevant commit.

d) Provide the link to this github repo below

<https://github.com/Mohitsai/polynomial>

## Exercise

Fork the course repo. Clone that fork locally. Ensure there is a remote called origin pointing to your fork and add a remote called upstream pointing to the course repo. Create a new branch called "worksheet\_01". Checkout this new branch. In the `student_notes` folder, create a new file called

`<your_last_name>_<your_first_name>_worksheet_01.txt` . In this file, answer the following question:

A friend presents you with a coin they claim to be fair. You flip the coin 5 times and it lands on Heads every single time. You flip the coin another 5 times, same result. How many times must this happen for you to start doubting the fairness of the coin? Explain your reasoning a bit.

add and commit this change with the message "contributing class notes". Push the changes to the origin/worksheet\_01 branch. Create a Pull Request against the course repo from this branch on github. Provide a link below to this PR.

Go back to the main branch so you can repeat this process in future worksheets or when you want to add any class notes for extra credit.

<https://github.com/gallettilance/Data-Science-Fundamentals/pull/451>