Subject Name: **Source Code Management**

Subject Code: **CS181**

Cluster: **Beta**

Department: **CSE**

****

|  |  |  |
| --- | --- | --- |
| **Submitted By:**  Aayush Bansal  2110990026  G1 |  | **Submitted To:**  Dr. Monit Kapoor |

List of Programs

|  |  |  |
| --- | --- | --- |
| S No. | Aim | Page No. |
| 1. | Setting up Git Client | 3-5 |
| 2. | Setting up Github Account | 6-7 |
| 3. | Program to Generate logs | 7-8 |
| 4. | Create and Visualize Branches. | 8-9 |
| 5. | Git Lifecycle and Descryption | 9-11 |

**Experiment No. 01**

**Aim:** Setting up of Git Client

**Installing and Configuring the Git client**

The following sections lists the steps required to install and configure the Git client - Git Bash and Git GUI – on computer.

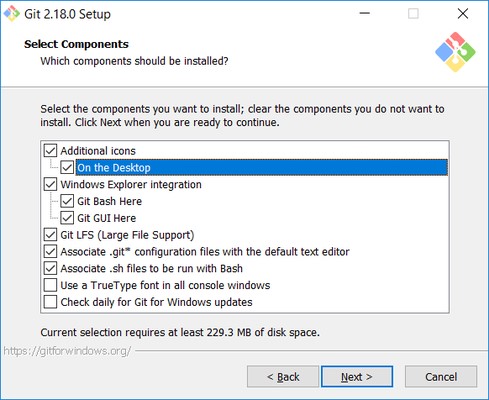
The remaining instructions here are specific for the Windows installation.

# 1. Git installation

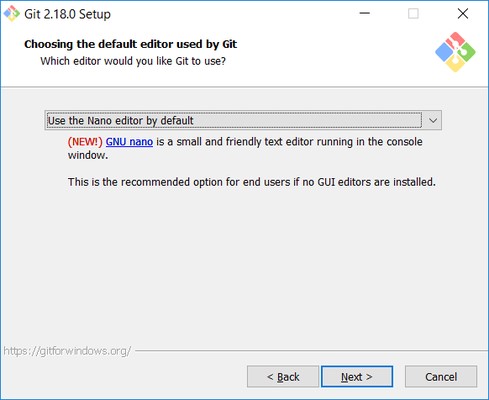
Download the Git program from [http://git-scm.com/downloads.](http://git-scm.com/downloads)

When running the installer, various screens appear (Windows screens shown). Generally, you can accept the default selections, *except in the screens below where you do not want the default selections:*

In the Select Componentsscreen, make sure Windows ExplorerIntegrationis selected as shown:



In the Choosing the default editor used by Gitdialog, it is strongly recommended that you DO NOT select the default VIM editor ,there are better modern editors available. Instead, choose Notepad++ or Nanoas they are much easier to use.



In the Adjusting your PATH screen, all three options are acceptable:

1. **Use Git from Git Bash only**: no integration, and no extra commands in your command path
2. **Use Git from the Windows Command Prompt**: adds flexibility - you can simply run git from a Windows command prompt, and is often the setting for people in industry - but this does add some extra commands.
3. **Use Git and optional Unix tools from the Windows Command Prompt**: this is also a robust choice and useful if you like to use Unix commands like grep.

**Experiment No. 02**

**Aim:** Setting up GitHub Account

This guide will walk you through setting up your GitHub account and getting started with GitHub's features for collaboration and community.

**Part 1: Configuring your GitHub account**

The first steps in starting with GitHub are to create an account, choose a product that fits your needs best, verify your email, set up two-factor authentication, and view your profile.

There are several types of accounts on GitHub. Every person who uses GitHub has their own user account, which can be part of multiple organizations and teams. Your user account is your identity on GitHub.com and represents you as an individual.

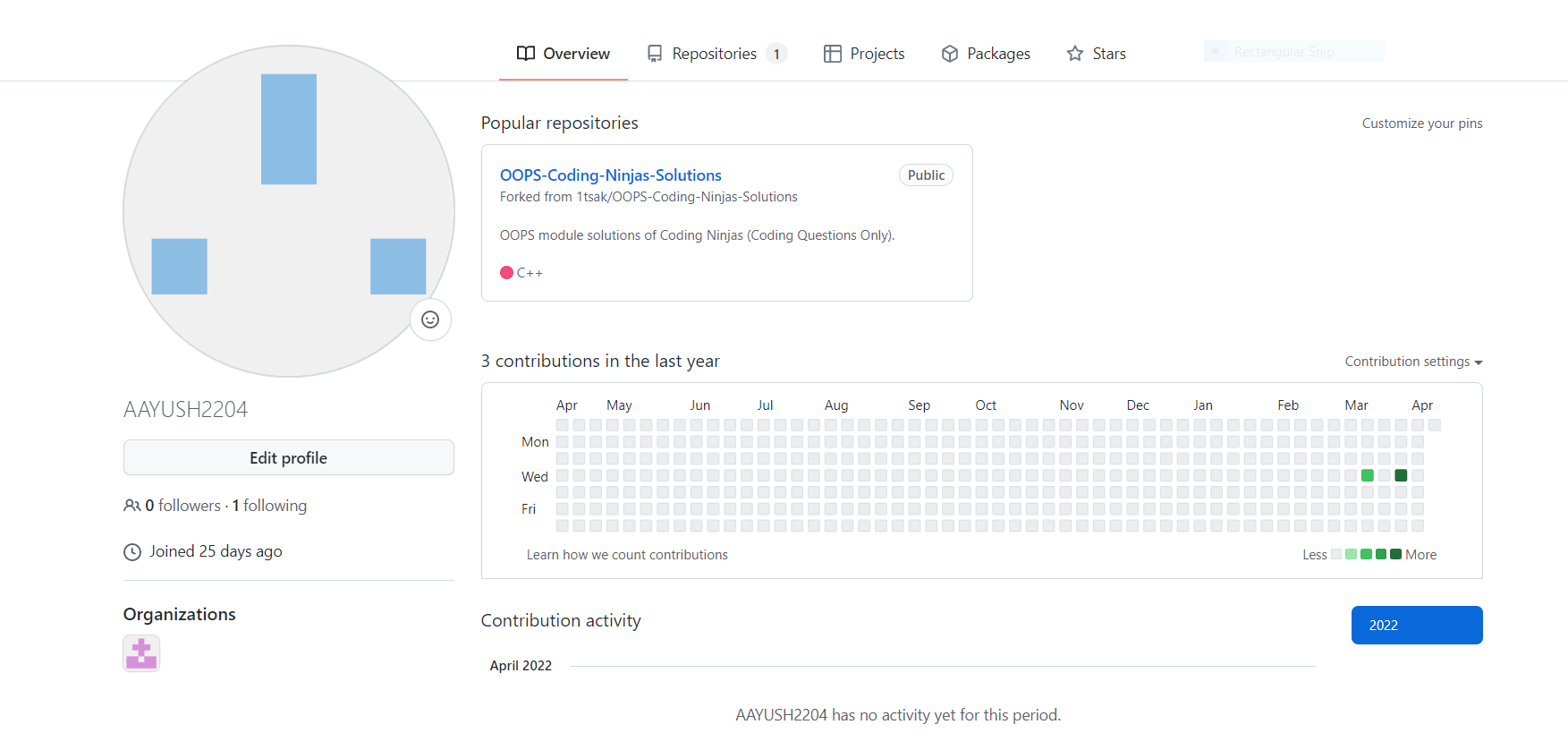
**Creating an account**

1. To sign up for an account on GitHub.com, navigate to <https://github.com/>

2. Now add our E-Mail ID to your Github account.

3. Sign Up into your account.

4. Then manage your github profile.

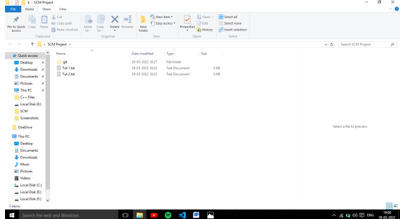


**Experiment No. 03**

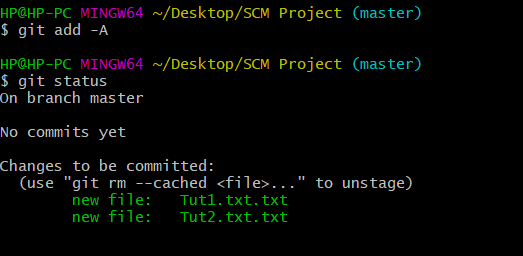
**Aim:** Program to Generate logs.

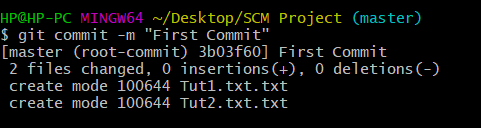
Git Log is a utility tool to review and read a history of everything that happens to a repository.Multiple options can be used with a git log to make history more specific.

Step 1. Firstly create a file in our repo which is untracked.

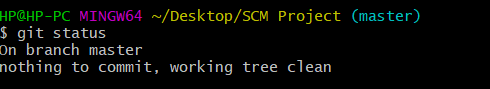


Step 2. We want to track our files, so firstly we add these files to the staging area by git add –a command and after adding the file we commit the file with the help of git commit –m “First Commit” command.

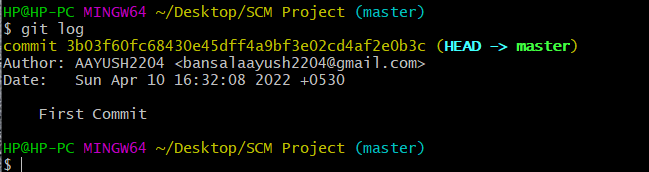




Step 3. After commiting our both files are tracked and we are working clean which can be seen from git status command.



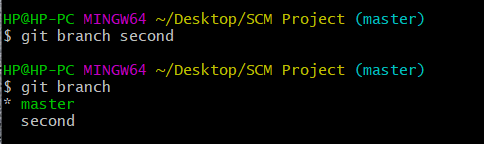
Step 4. After that we can see our changes with the help of git log command.



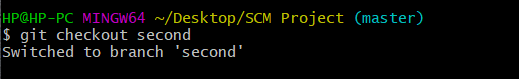
**Experiment No. 04**

**Aim:** Create and visualize branches

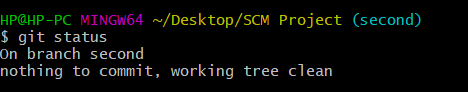
Step 1. We know that our files are commited in master branch initially.Now,we will create another branch Second with the git branch command.



Step 2. We can shift to our new branch by git checkout command easily.



Step 3. We can easily make changes In our new branch which will not be implemented in the master branch and if we want to go back to the previous project then we can shift back to the master branch by the same command.

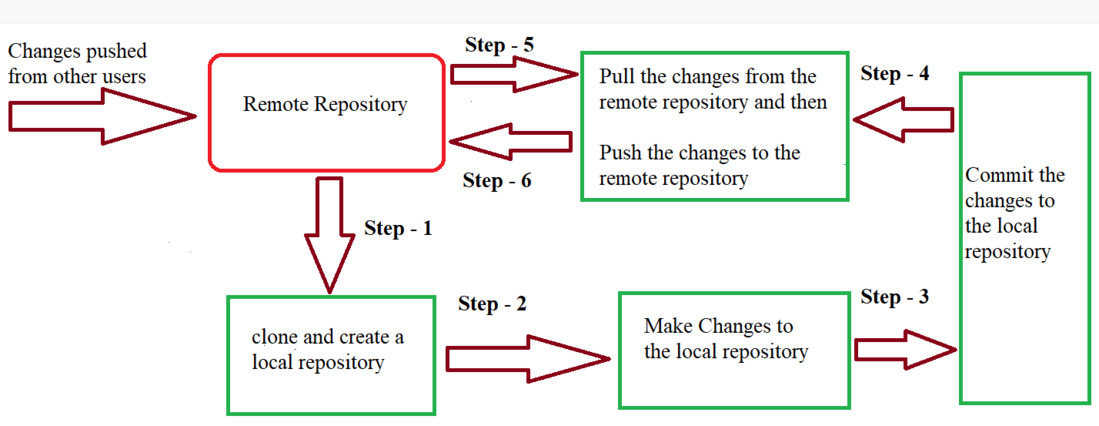
s

**Experiment No. 05**

**Aim:** Git lifecycle description

**Git – Life Cycle**

Git is used in our day-to-day work, we use git for keeping a track of our files, working in a collaboration with our team, to go back to our previous code versions if we face some error. Git helps us in many ways. Let us look at the Life Cycle that git has and understand more about its life cycle. Let us see some of the basic steps that we follow while working with Git –



When a directory is made a git repository, there are mainly 3 states which make the essence of Git Version Control System. The three states are –

* Working Directory
* Staging Area
* Git Directory

Let us understand in detail about each state.

**1. Working Directory**

Whenever we want to initialize our local project directory to make it a git repository, we use the ***git init*** command. After this command, git becomes aware of the files in the project although it doesn’t track the files yet. The files are further tracked in the staging area.

*git init*

**2. Staging Area**

Now, to track the different versions of our files we use the command ***git add***. We can term a staging area as a place where different versions of our files are stored. ***git add*** command copies the version of your file from your working directory to the staging area. We can, however, choose which files we need to add to the staging area because in our working directory there are some files that we don’t want to get tracked, examples include node modules, env files, temporary files, etc. Indexing in Git is the one that helps Git in understanding which files need to be added or sent. You can find your staging area in the ***.git*** folder inside the ***index*** file.

**3. Git Directory**

Now since we have all the files that are to be tracked and are ready in the staging area, we are ready to commit our files using the ***git commit***command. Commit helps us in keeping the track of the metadata of the files in our staging area. We specify every commit with a message which tells what the commit is about. Git preserves the information or the metadata of the files that were committed in a Git Directory which helps Git in tracking files and basically it preserves the photocopy of the committed files. Commit also stores the name of the author who did the commit, files that are committed, and the date at which they are committed along with the commit message.