**GIT**

**What is Git?**

Git is a powerful and popular version control system that enables effective tracking of changes in source code. It was developed by Linus Torvalds in 2005 for Linux kernel development and It is used for keeping track of code changes and collaborating with others on code. It uses a decentralized model where each developer has their own copy of the repository and works immediately on the project. Git manages the projects with repositories and can clone a project to operate locally on it. With staging and committing it track changes and control. You can pull the latest code of the project to the local copy, and push local updates to the main projects.

**Why Use Git?**

You need to know that around 70% of developers worldwide use Git for development. Some of the prominent reasons for using Git are:

• Developers can work together from anywhere.

• Developers can see the full history and can compare the previous and new changes of the project.

• Developers can retreat to earlier versions of a project.

**Working with Git**

When a folder is initialised with Git, it becomes a repository—a special location where Git logs all changes made to a hidden folder. In that folder, each time you change, add, or remove a file, Git takes note of the change and marks the file as “modified.” You can choose which modified files you want to save by staging them, so don’t worry. Consider staging as getting the changes ready for a particular snapshot that you want to keep. Once the staged changes are to your satisfaction, commit them, and Git will keep a permanent copy of those files in its history.

Git is great because it maintains a complete record of each commit you make.

**Version Control Systems**

**What is a “version control system”?**

Version control systems are a category of software tools that helps in recording changes made to files by keeping a track of modifications done in the code.

**Why Version Control system is so Important?**

As we know that a software product is developed in collaboration by a group of developers they might be located at different locations and each one of them contributes to some specific kind of functionality/features. So in order to contribute to the product, they made modifications to the source code(either by adding or removing).

A version control system is a kind of software that helps the developer team to efficiently communicate and manage(track) all the changes that have been made to the source code along with the information like who made and what changes have been made. A separate branch is created for every contributor who made the changes and the changes aren’t merged into the original source code unless all are analyzed as soon as the changes are green signaled they merged to the main source code. It not only keeps source code organized but also improves productivity by making the development process smooth.

**Benefits of the version control system:**

• Enhances the project development speed by providing efficient collaboration,

• Leverages the productivity, expedites product delivery, and skills of the employees through better communication and assistance,

• Reduce possibilities of errors and conflicts meanwhile project development through traceability to every small change,

• Employees or contributors of the project can contribute from anywhere irrespective of the different geographical locations through this VCS,

• For each different contributor to the project, a different working copy is maintained and not merged to the main file unless the working copy is validated. The most popular example is Git, Helix core, Microsoft TFS,

• Helps in recovery in case of any disaster or contingent situation,

• Informs us about Who, What, When, Why changes have been made.

**Types of Version Control Systems:**

• Local Version Control Systems

• Centralized Version Control Systems

• Distributed Version Control Systems

**Local Version Control Systems:**

It is one of the simplest forms and has a database that kept all the changes to files under revision control. RCS is one of the most common VCS tools. It keeps patch sets (differences between files) in a special format on disk. By adding up all the patches it can then re-create what any file looked like at any point in time.

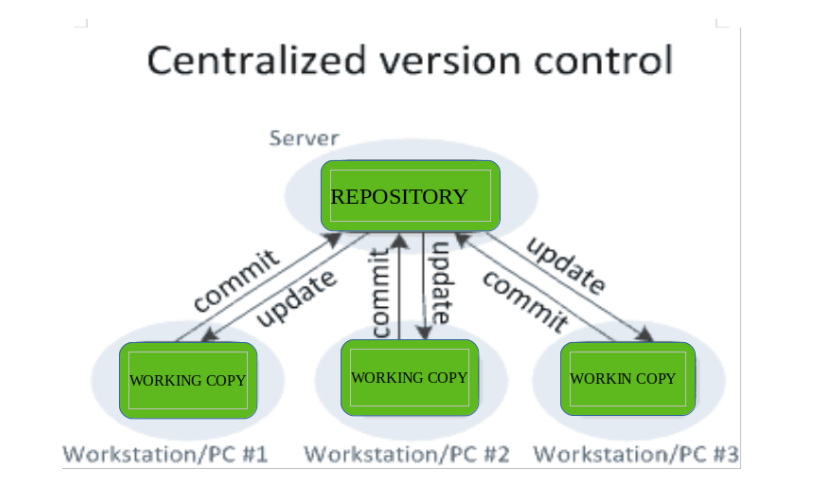
**Centralized Version Control Systems:**

Centralized version control systems contain just one repository globally and every user need to commit for reflecting one’s changes in the repository. It is possible for others to see your changes by updating.

Two things are required to make your changes visible to others which are:

• You commit

• They update



The benefit of CVCS (Centralized Version Control Systems) makes collaboration amongst developers along with providing an insight to a certain extent on what everyone else is doing on the project. It allows administrators to fine-grained control over who can do what.

It has some downsides as well which led to the development of DVS. The most obvious is the single point of failure that the centralized repository represents if it goes down during that period collaboration and saving versioned changes is not possible. What if the hard disk of the central database becomes corrupted, and proper backups haven’t been kept? You lose absolutely everything.

**Distributed Version Control Systems:**

Distributed version control systems contain multiple repositories. Each user has their own repository and working copy. Just committing your changes will not give others access to your changes. This is because commit will reflect those changes in your local repository and you need to push them in order to make them visible on the central repository. Similarly, When you update, you do not get others’ changes unless you have first pulled those changes into your repository.

To make your changes visible to others, 4 things are required:

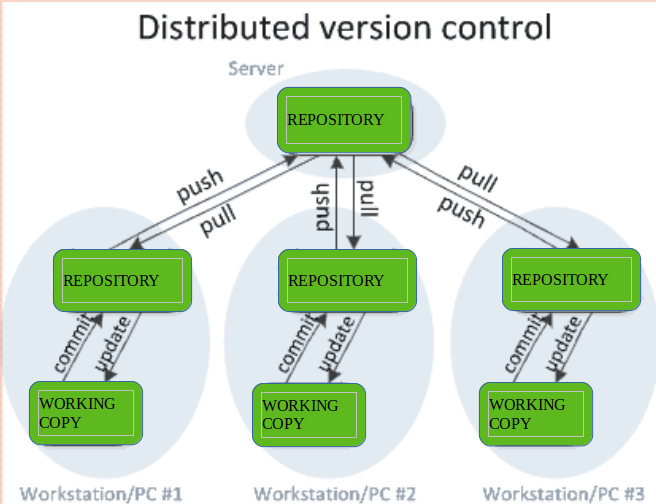
• You commit

• You push

• They pull

• They update

The most popular distributed version control systems are Git, and Mercurial. They help us overcome the problem of single point of failure.



**Purpose of Version Control:**

• Multiple people can work simultaneously on a single project. Everyone works on and edits their own copy of the files and it is up to them when they wish to share the changes made by them with the rest of the team.

• It also enables one person to use multiple computers to work on a project, so it is valuable even if you are working by yourself.

• It integrates the work that is done simultaneously by different members of the team. In some rare cases, when conflicting edits are made by two people to the same line of a file, then human assistance is requested by the version control system in deciding what should be done.

• Version control provides access to the historical versions of a project. This is insurance against computer crashes or data loss. If any mistake is made, you can easily roll back to a previous version. It is also possible to undo specific edits that too without losing the work done in the meanwhile. It can be easily known when, why, and by whom any part of a file was edited.

**Installation of Git**

Here are the steps to set up Git on your Windows machine:

1. Download Git:

Visit the official Git website: <https://git-scm.com/download/win>.

• Click the download link for the latest version for Windows.

• The download will start automatically. Once it's finished, run the installer.

2. Run the Installer:

• Locate the downloaded executable file (e.g., Git-2.30.0-64-bit.exe).

• Double-click the file to run the installer.

3. After Installing the git, we can customize it’s environment accordingly. The customization shall be done on any given computer. Git comes with a tool called git config that helps to set configuration variables, that look after the operation of git. In order to set these configuration values as global, add the –global option, and if you omit the –global option, then your configurations are specific for the current Git repository. Git can store configuration variables in three different following files:

• /etc/gitconfig: It is a file that contains configuration for every user and repository on the system. Since it is system-wide configuration file, to configure these values one must have administrative permission to make changes. –system option can be used.

• ~/.gitconfig: This file contains user-specific values. –global option can be used and all the repositories on the system can be configured.

• config file(current repository): It is specific to a single repository (current). –local option can be used for making configurations.(If no option is used then it is by default –local).

4. Complete the Installation:

Click "Next" to continue.

The installer will start copying files and complete the installation.

5. Verify the Installation:

Open the Command Prompt (or PowerShell) and type git --version and press Enter. You should see the installed Git version.

6. Configure Git:

Before you start using Git, you should configure your name and email. Open the Command Prompt and run the following commands, replacing "Your Name" and "your.email@example.com" with your name and email:

**git config --global user.name "Your Name"**

**git config --global user.email** [**your.email@example.com**](mailto:your.email@example.com)

7. (Optional): Set Up SSH Keys (for secure authentication):

If you plan to use Git with remote repositories and want to use SSH for secure authentication, you can generate SSH keys and add your public key to your Git hosting service (e.g., GitHub, GitLab). Here are the basic steps:

* Open the Command Prompt.
* Run the following command to generate an SSH key pair:

**ssh-keygen -t rsa -b 4096 -C "your.email@example.com"**

* Press Enter to accept the default file location (C:\Users\<your\_username>\.ssh\id\_rsa) and choose a strong passphrase for added security.
* Add your SSH key to the SSH agent using ssh-add ~/.ssh/id\_rsa.
* Copy your public key from ~/.ssh/id\_rsa.pub and add it to your Git hosting service's SSH key settings.

Now, Git is installed and configured on your Windows system, and you can start using it for version control.

**What is GitHub about?**

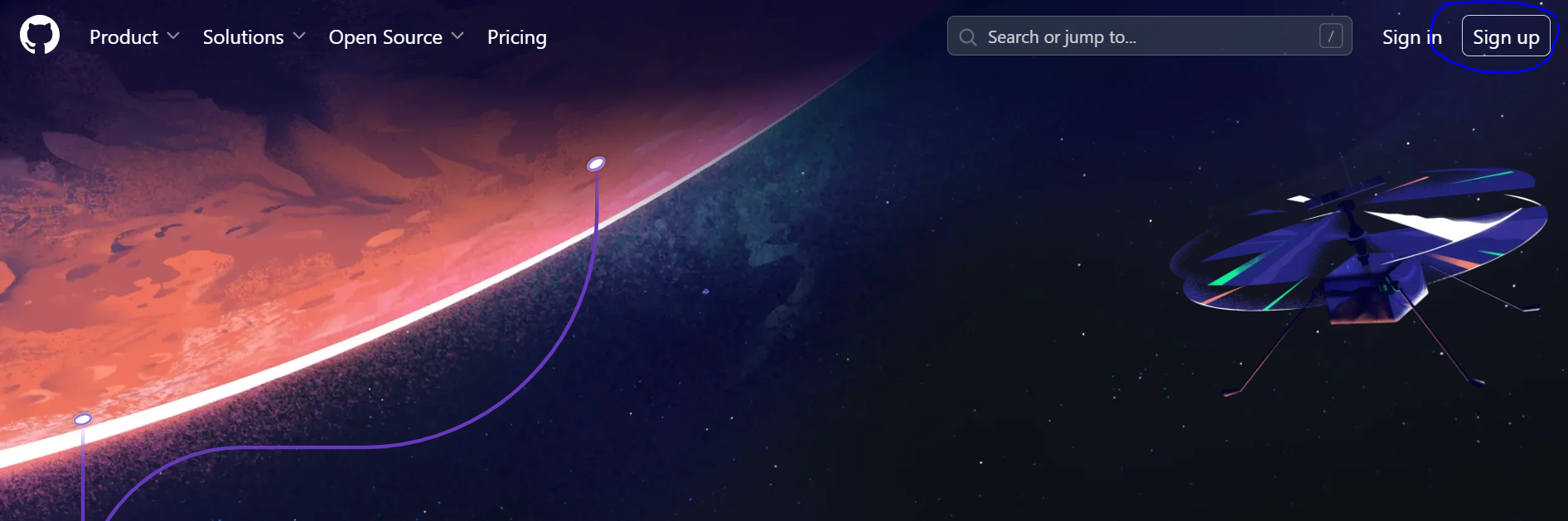
Github is an online platform where we can share our codes(or projects) online hassle-free. Github is place where we host our local git repository online. it basically allow you to work collaboratively within a group of peoples. It hosts all the features of Git and have its own too.

In simple words, Github might be considered as a social media which is made for the developers where they share their work. it might be any project regarding website development or any design of website, or some operating systems like Android, or Linux, etc.

Github is owned by Microsoft, provides access to public(free) and private(paid) repositories. It allows free repository to host files up to 100mb and total size for 2GB.

A. Creating a GitHub Account

Step 1: Go to github.com and enter the required user credentials asked on the site and then click on the SignUp for GitHub button.



B. Creating a new Repository

• Login to your Github account

• On the dashboard click on the Green Button starting New repository.

• Make sure to verify the Github account by going into the mail which was provided when creating the account.

• Once verification has been done, the following screen comes

C. Start by giving a repository name, description(optional) and select the visibility and accessibility mode for the repository

D. Click on Create repository

**Top 12 Most Used Git Commands:**

1) git config

Before you can start using Git, you need to configure it. This command allows you to specify the username and email address that will be used with your commits.

#sets up Git with your name

*git config --global user.name "<Your-Full-Name>"*

# sets up Git with your email

*git config --global user.email "<your-email-address>"*

2) git init

A git repository must first be created before you can make commits or do anything else with it. We’ll use the git init command to create a new Git repository. The init subcommand stands for “initialise,” which is useful because it’s the command that handles all of a repository’s initial setup. In a moment, we’ll look at what it does. The git init command creates all of the necessary files and directories for Git to keep track of everything

*git init*

3) git clone

The command we’ll be doing on the terminal is git clone, which is how cloning is related to Git. The git clone command takes a path (typically a URL) to the Git repository you want to clone. The command is git clone, and the path to the Git repository you want to clone is passed as an argument. This is the URL for the project we’ll be working on throughout the course. git clone produces a local working copy of a remote repository’s source code. When you clone a repository, the code is downloaded to your machine automatically. If you have permission, this command will add the original location as a remote location, allowing you to grab changes from it and push changes to it.

*git clone https://github.com/<repo-url>*

4) git status

The git status command is our key to Git’s mind. It will inform us of Git’s thoughts and the state of our repository as seen by Git. When you’re first getting started, you should always use the git status command! Seriously. It’s a good idea to start running it after any other command. This will assist you in learning Git and avoiding making (potentially) inaccurate assumptions about the state of your files/repository.

*git status*

Depending on the state of your files, working directory, and repository, the git status tool will display a lot of information.

*git add*

To move files from the Working Directory to the Staging Index, use the git add command. The git add command saves your changes in a file to the staging area, allowing you to compare your local version to the remote repository’s version. Before committing your new or changed file, use the git add command to add it to the staging area. To add specific files:

*git add <file1> <file2> … <fileN>*

To add all the files:

*git add .*

6) git commit

This command saves a log message along with the commit id of the modifications made to the git repository. The modifications are saved in your local repository with git commit. You must include a brief description of the changes made every time you commit your code changes. This commit message aids others in comprehending the changes made.

*git commit –m "<Type your commit message here>"*

7) git push

This command pushes the contents of your local repository to the remote repository you’ve added. This pushes your master branch’s commits to the newly added remote repository. If a named branch does not exist in the remote repository, it will be created.

*git push*

8) git branch

Add a new branch to an existing branch, list all existing branches, and delete a branch with git branch. This command is used to perform operations on a branch that has been specified. When you run this command, Git will remove all files and folders from the Working Directory it is tracking (files that Git tracks are stored in the repository, so nothing is lost) and pull all of the files and directories from the commit that the branch links to from the repository. Create a new branch locally:

*git branch <branch-name>*

Take a look at the branches and check out on which branch you’re currently working:

*git branch* or *git branch --list*

9) git checkout

We can use the git checkout command to switch to an existing branch or to create and switch to a new one. To accomplish this, the branch you want to switch to must be present in your local system, and any modifications made to your current branch must be committed or stashed before switching. This command can also be used to check out files. When you run this command, Git will remove all files and folders from the Working Directory it is tracking (files that Git tracks are stored in the repository, so nothing is lost) and pull all of the files and directories from the commit that the branch links to from the repository.

*git checkout <branch-name>*

10) git merge

The history of the specified branch is merged into the current branch with this command. The command git merge joins your branch to the parent branch. Depending on your process, the parent branch can be either a development or a master branch. If there are no conflicts, it will make a new commit automatically. You should be on the branch you want to merge with your parent branch before running the git merge command. The history of the specified branch is merged into the current branch with this command.

*git merge <name-of-branch-to-merge-in>*

11) git pull

The contents of the remote repository are fetched and integrated into your local repository using this command. git pull pulls the most recent changes from the remote server into the local repository, ensuring you have the most up-to-date information from your coworkers.

*git pull*

12) git log

The git log command is used to show all of a repository’s commits. This command displays a log of all commits made to the current branch so far.

*git log*

**Git Command used during the session:**

git config --global user.name "abc1120"

git config --global user.email [abc@in.ey.com](mailto:abc@in.ey.com)

git clone <https://github.com/ps1120/Python_Training.git>

cd Python\_Training

git pull origin main

git branch

git status

git checkout -b <branch-name>

git branch

git status

git add .

git commit -m "<commit message>"

git status

git push origin HEAD:<branch-name>

git checkout main

git branch

**PEP-8**

Reads: <https://peps.python.org/pep-0008/>

PEP 8 – Style Guide for Python Code

One of Guido’s key insights is that code is read much more often than it is written. The guidelines provided here are intended to improve the readability of code and make it consistent across the wide spectrum of Python code. As PEP 20 says, “Readability counts”.