Git and GitHub

1. Introduction

What is Git?

- Git is a **distributed version control system (DVCS)**.
- Developed by **Linus Torvalds** in 2005 (creator of Linux kernel).
- Helps developers **track changes**, **collaborate** on code, and **manage versions** efficiently.
- Works **offline and locally**, with the ability to **sync with remote repositories** when needed.

What is GitHub?

- GitHub is a **web-based hosting service** for Git repositories.
- Provides a graphical interface, collaboration tools, issue tracking, CI/CD integration, and more.
- Founded in 2008, acquired by Microsoft in 2018.
- Offers **public and private repositories**, pull requests, actions (CI/CD), discussions, etc.

2. How Git Works (Core Concepts)

Git Lifecycle (Three Stages)

1. **Working Directory** – Local files being edited.
2. **Staging Area (Index)** – Where changes are prepared before committing.
3. **Repository (.git folder)** – Stores committed changes.

Git Basic Commands

Command	**Description**
`git init`	Initializes a new Git repo
`git clone <url>`	Copies a remote repo
`git add <file>`	Adds changes to staging
`git commit -m "msg"`	Commits staged changes

Command	Description
`git status`	Shows current status
`git push`	Uploads changes to remote
`git pull`	Downloads and merges changes from remote
`git log`	Shows commit history
`git branch`	Manages branches

Branching and Merging

- **Branch** – A separate line of development.
- `git branch <name>` – Creates a branch.
- `git checkout <name>` – Switches to a branch.
- `git merge <branch>` – Merges another branch into the current one.

3. GitHub Features and Workflow

Key Concepts

Concept	Description
Repository	Stores code/project
Fork	Copy of another repo (to propose changes)
Pull Request (PR)	Request to merge code changes
Issues	Bug/feature tracking
Actions	CI/CD pipelines
Wiki	Project documentation
Discussions	Forum-style conversations
Releases	Packaged versions of your software

Collaboration Workflow

Fork → Clone → Create Branch → Make Changes → Push → Pull Request → Review → Merge

4. Use Cases of Git & GitHub

Git

- Version control for any file type.
- Offline code tracking and recovery.
- Individual or collaborative development.
- Open-source and enterprise software management.

GitHub

- Hosting open-source/public code.
- Private project management.
- Team collaboration (pull requests, code reviews).
- CI/CD integration (GitHub Actions).
- Deployment (e.g., GitHub Pages for websites).
- Educational & portfolio use (profile README, contributions graph).

5. Advantages

Git

- Fast and lightweight.
- Fully distributed.
- Complete history and rollback.
- Branching and merging is easy.
- Works offline.

GitHub

- Cloud-based and accessible.
- User-friendly interface.
- Powerful integrations (Slack, VS Code, Jenkins, etc.).
- Promotes open-source contributions.
- Built-in tools: Actions, Pages, Issues, PRs.

6. Disadvantages

Git

- Steep learning curve for beginners.
- Complex commands for some operations (e.g., rebase, cherry-pick).
- Merge conflicts can be hard to resolve.

GitHub

- Private repos have limits on free plans.
- Owned by Microsoft (some open-source advocates prefer independent alternatives).
- Dependency on internet connection.

7. Alternatives

Git Alternatives (Version Control Systems)

Tool	**Description**
Mercurial (hg)	Simpler than Git, also distributed
Subversion (SVN)	Centralized version control
Bazaar	Distributed, Python-based
Perforce	Enterprise-level, centralized

GitHub Alternatives (Hosting Services)

Tool	**Description**
GitLab	Self-hosted and cloud-hosted options; DevOps tools
Bitbucket	Atlassian product with Jira integration
SourceForge	Older but still used in open-source
Azure DevOps	Microsoft’s enterprise code hosting

Tool	Description
Gitea	Lightweight, self-hosted Git service

8. Real-World Applications

- **Software development teams** (code versioning, branching, collaboration).
- **Open-source contributors** (pull requests, forks).
- **DevOps pipelines** (CI/CD with GitHub Actions).
- **Documentation management** (Markdown files in repos).
- **Research projects** (versioning LaTeX or Jupyter Notebooks).
- **Web developers** (hosting static websites using GitHub Pages).
- **Students and portfolios** (GitHub profiles as resumes).

9. Best Practices

- Commit frequently with clear messages.
- Use `.gitignore` to avoid uploading unnecessary files.
- Work in feature branches.
- Regularly pull from upstream repo.
- Write meaningful pull request descriptions.
- Tag releases for versioning.
- Use GitHub Issues for tracking bugs/tasks.

Git & GitHub Installation and Setup

PART A: Setting Up GitHub (Using Gmail)

Step 1: Log In to Gmail

1. Open a web browser (e.g., Chrome, Edge).
2. Go to https://mail.google.com.
3. Enter your Gmail ID and password.
4. Complete 2FA (if enabled).

Done? Now move to GitHub.

Step 2: Create a GitHub Account

1. Visit https://github.com.
2. Click on **Sign Up** (top-right corner).
3. Enter your Gmail email address.
4. Create a **username** (e.g., rohanraj-dev).
5. Create a **strong password**.
6. GitHub will ask you to:
    - Solve a puzzle (to verify you're human).
    - Choose **free plan** (for most users).
    - Skip/choose developer preferences.
7. Confirm your Gmail email address (open Gmail → click GitHub verification email).

Now you have a GitHub account!

Step 3: Set Up GitHub Profile

1. Add your **profile picture**, **bio**, **location**, and **social links** (optional).
2. Create a **README profile** (optional, to show up on your profile page):
    - Create a new repository with the exact name as your username.
    - Add a `README.md` file with introduction, skills, etc.

PART B: Installing Git on Your Computer

Step 4: Download Git

For Windows:

1. Go to https://git-scm.com/download/win
2. The `.exe` file will download automatically.
3. Open the file and run the **Git Setup Wizard**.

Step 5: Install Git (for Windows - Detailed)

During installation, keep default settings unless you know what you're doing. Notable steps:

Step	**Option**	**Recommended**
Select components	All checked	✅
Text editor	Choose **Visual Studio Code** if available	✅
Adjust PATH	“Git from the command line and also from 3rd-party software”	✅
Line endings	Checkout Windows-style, commit Unix-style	✅
Terminal emulator	Use MinTTY (default)	✅
Configuring extras	Enable file system caching, Enable Git Credential Manager	✅

Click **Finish** at the end.

PART C: Git Configuration (First Time Setup)

Step 6: Open Git Bash or Terminal

- On Windows, search and open **Git Bash**.

Step 7: Configure Git with Your GitHub Credentials
git config --global user.name "Your Full Name"

git config --global user.email "youremail@gmail.com"

Example:

git config --global user.name "Rohan Raj Poudel"

git config --global user.email "rohanraj123@gmail.com"

Step 8: Confirm Configuration
git config --list

Output will show your name and email.

PART D: Connecting Git with GitHub

Step 9: SSH with Git & GitHub + 💻 VS Code Integration (Full Guide)

PART A: What is SSH and Why Use It?

What is SSH?

- **SSH (Secure Shell)** is a cryptographic network protocol to connect and authenticate securely between systems.

- In Git/GitHub context, SSH is used to:
    - **Push and pull** code securely.
    - Authenticate **without entering your username/password** each time.

Why SSH over HTTPS?

SSH	**HTTPS**
Authentication via key (fast)	Authentication via username/password or token (slower)
Works well for devs	Preferred for automation scripts
Secure & passwordless	May require token/prompt on each push

PART B: Generate SSH Key

Step 1: Open Terminal / Git Bash

On Windows, open **Git Bash** On Mac/Linux, open **Terminal**

Step 2: Generate SSH Key

ssh-keygen -t ed25519 -C "your_email@example.com"

- `-t ed25519` = modern encryption method
- `-C` is a label (your GitHub email)

Press Enter to:

- Accept default path (`~/.ssh/id_ed25519`)
- Optionally, add a passphrase (you can leave it blank)

This creates:

- `~/.ssh/id_ed25519` (private key — **keep safe**)
- `~/.ssh/id_ed25519.pub` (public key — **share with GitHub**)

Step 3: Start SSH Agent and Add Your Key

eval "$(ssh-agent -s)"

ssh-add ~/.ssh/id_ed25519

This loads your key into the session.

Step 4: Add SSH Key to GitHub

1. Get the public key:

cat ~/.ssh/id_ed25519.pub

2. Copy the output (starts with `ssh-ed25519`).

3. Go to **GitHub > Settings > SSH and GPG Keys** Link: https://github.com/settings/keys

4. Click **New SSH Key**.
    - Title: "My Laptop" (or anything)
    - Paste the copied key in the Key field.
    - Click **Add SSH Key**

You've now authorized your PC to communicate with GitHub via SSH.

Step 5: Test SSH Connection

Run:

ssh -T git@github.com

Expected output:

Hi your-username! You've successfully authenticated, but GitHub does not provide shell access.

If you see this, **SSH is working properly**.

PART C: Use SSH with Git Repositories

Cloning a Repo via SSH

1. On GitHub, go to any repository.

2. Click the green **Code** button.

3. Choose **SSH** tab and copy the link: Example:
   `git@github.com:username/repo-name.git`

4. In terminal:

git clone git@github.com:username/repo-name.git

PART D: VS Code Integration with Git & GitHub

Step 1: Open VS Code & Enable Git

1. Open your project folder.

2. VS Code auto-detects Git if the folder is a Git repo (`.git` exists).

3. Use the **Source Control** icon (left sidebar) to view:
    - Unstaged/staged changes
    - Commit history
    - Branch info

Step 4: Use VS Code Terminal for Git SSH

You can run Git commands directly from:

- **VS Code Terminal**: `Ctrl + `` (backtick)

- All Git commands (clone, push, pull) work the same.

- Git will use your SSH key by default if:
    - It's loaded in SSH agent
    - You clone using the SSH URL

Step 5: GitHub Authentication in VS Code

For full GitHub integration:

1. Install extension: **GitHub Pull Requests and Issues**

2. Sign in when prompted.

3. You can now:
    - Create pull requests from inside VS Code
    - View GitHub Issues
    - Review and merge code

PART E: Typical SSH Git Workflow in VS Code

1. **Clone via SSH**:

   git clone git@github.com:yourusername/yourrepo.git

2. **Open folder in VS Code**.

3. Make code changes.

4. Stage changes (via Source Control or terminal):

   git add .

   git commit -m "Your commit message"

   git push origin main

All communication is secured over SSH.

PART E: Test Everything

Step 11: Create a Test Repository on GitHub

1. Go to https://github.com.
2. Click **+** (top-right) → **New repository**.
3. Name it (e.g., `my-first-repo`).
4. Add README file (optional).
5. Click **Create Repository**.

Step 12: Clone Repo Locally

git clone git@github.com:yourusername/my-first-repo.git
cd my-first-repo
OR (if using HTTPS):
git clone https://github.com/yourusername/my-first-repo.git
cd my-first-repo
You now have a GitHub repo cloned on your PC.