

Git Basics

What is Version Control System?

- A version control system (VCS) records changes to files over time.
- Allows multiple people to work on a project simultaneously.
- Maintains a history of changes, making it easy to revert to previous versions.
- Examples: Git, Subversion (SVN), Mercurial.

What is git?

- Git is a distributed version control system.
- It helps developers track changes in source code during software development.
- Created by Linus Torvalds in 2005.
- Enables collaboration and management of code history.

Benefits of git

Distributed Development :

Every developer has a full copy of the repository.

Collaboration :

Multiple people can work on the same project without conflicts.

Branching and Merging :

Easily create, switch, and merge branches for new features or bug fixes

History and Traceability :

Track who made changes, when, and why.

Performance : Fast operations for most tasks due to local repositories.

Open Source : Free to use and widely supported.

Installation

Download **Visual Studio Code** from below link

<https://code.visualstudio.com/download>

Download **git** from below

<https://git-scm.com/downloads>

- After installation of git, open cmd prompt from your windows laptop and type **git**
- If you are getting the below output then installation is successful

```
(base) C:\Users\medsi>git
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
      [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
      [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--no-lazy-fetch]
      [--no-optional-locks] [--no-advice] [--bare] [--git-dir=<path>]
      [--work-tree=<path>] [--namespace=<name>] [--config-env=<name>=<envvar>]
      <command> [<args>]
```

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)

clone	Clone a repository into a new directory
init	Create an empty Git repository or reinitialize an existing one

Repository Setup

- Create a repository in GitHub named `git-practice` (add a ReadMe)
- Copy the repository link and clone it locally:
`git clone <repository-link>`
- Switch to the new folder:
`cd git-practice` (in VS Code terminal)

Initial Branch Setup

- Check the current branch:
`git branch`
- Pull the latest changes:
`git pull`
- Create a new branch from the main branch:
`git checkout -b develop main` # `git checkout -b new-branch old-branch`
- Verify branches and status:
`git branch`
`git status`

Adding the first function

- Create `sample.py` and add the `addition` function.
- Check status:
`git status`
- Add files:
`git add .` # `git add file1.py file2.py ...` To add specific files
- Commit changes:
`git commit -m "added addition function"`
- Push to remote (set upstream if needed):
`git push`
- If error:
`git push --set-upstream origin develop`

As, `develop` branch is not there in repository, we need to set it as upstream

Feature Branch: Subtraction

- Create a feature branch from develop:

```
git checkout -b feature/subtraction develop
```

- Add the subtraction function in `sample.py`

- Add, commit, and push:

```
git add .
```

```
git commit -m "added subtraction function"
```

```
git push --set-upstream origin feature/subtraction
```

Feature Branch: Multiplication

- Create another feature branch from develop:

```
git checkout -b feature/multiplication develop
```

- Add the multiplication function in `sample.py`

- Add, commit, and push:

```
git add .
```

```
git commit -m "added multiplication function"
```

```
git push --set-upstream origin feature/multiplication
```

Merging and Pull Requests

- Merge feature/multiplication with develop:
 - Switch to feature/multiplication: `git switch feature/multiplication`
 - Merge: `git merge develop`
- Go to GitHub and raise a Pull Request (PR):
 - Base: develop ← Compare: feature/multiplication
- Switch to develop and pull latest changes:

```
git switch develop  
git pull
```


Handling Merge Conflicts

- Switch to feature/subtraction and merge develop:

```
git switch feature/subtraction
git merge develop #Remember latest develop changes to be pulled before
```

- If conflicts occur:

- Resolve conflicts by deleting lines starting with <<<<<<, =====, and >>>>>>.
- If needed, abort merge: `git merge --abort`
- Switch branches as needed, then retry merge.

- After resolving:

```
git add .
git commit -m "merged with develop branch"
git push
```

- Raise a PR to merge feature/subtraction with develop.

Implement CICD

- Switch to develop branch and create yml file in folder **.github/workflows/cicd.yml**
- Raise a PR to merge feature/subtraction with develop.

```
name: Python CICD Pipeline

# This defines the trigger
on:
  pull_request:
    branches:
      - develop # Trigger only when a PR is raised against develop

jobs:
  test-code:
    runs-on: ubuntu-latest # mention your virtual machine (OS) here

    steps:
      - name: Checkout code
        uses: actions/checkout@v4 # This checks out your code so that the workflow can access
it

      - name: Set up Python
        uses: actions/setup-python@v4 # This sets up the Python environment
        with:
          python-version: '3.12'

      - name: Run Python Script
        run: |
          # This simply runs your file to make sure there are no errors
          python sample.py
```

Working with Commit History

- Switch to develop and pull latest changes:

```
git switch develop
git pull
```

- To retrieve a previous version of `sample.py`:

- Find the commit hash (e.g., 793295e9a4eae6105b20920282351942c44a237)
- Run:

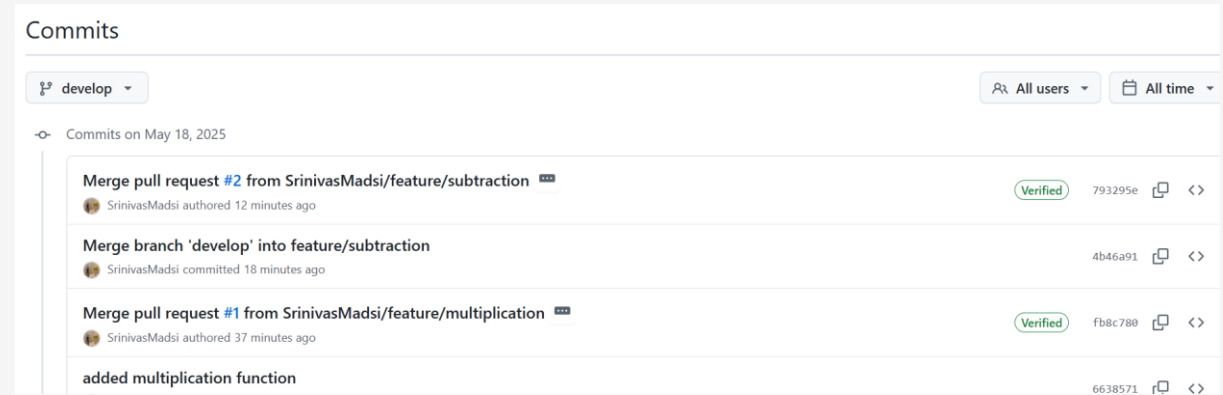
```
git log or directly from github like below
```

```
git checkout 793295e9a4eae6105b20920282351942c44a237
```

Or

```
git reset 793295e9a4eae6105b20920282351942c44a237
```

- Review changes; `commit` if needed, or use `git stash` to save changes temporarily.
- Check status: `git status`



Adding .env and .gitignore

- Create a new `.env` file and a `.gitignore` file.
- Add `.env` to `.gitignore` file
- Ensure both files are inside the `git-practice` folder.
- Add, commit, and push:

```
git add .  
git commit -m "added git ignore"  
git push
```

git Commands

- Clone a repository from github
`git clone <repository-link>`
- Check current branch:
`git branch`
- Create and switch to a new branch:
`git checkout -b <new-branch-name>`
`<base-branch-name>`
- Switch to an existing branch:
`git switch <branch-name>`
- Check status of files:
`git status`
- Pull latest changes from remote:
`git pull`
- Add all files to staging:
`git add .`
- Commit changes with a message:
`git commit -m "your message"`
- Push changes to remote branch:
`git push`

Add .gitignore file and list files/folders to ignore (e.g., .env)

- Push and set upstream for new branch:
`git push --set-upstream origin <branch-name>`
- Merge another branch into current branch:
`git merge <branch-name>`
- Abort a merge (if conflicts or issues):
`git merge -abort`
- After resolving conflicts, add and commit:
`git add .`
`git commit -m "merged with <branch-name>"`
- View commit logs:
`git log`
- Restore a file from a previous commit:
`git checkout <commit-hash> -- <filename>`
- Temporarily save uncommitted changes:
`git stash`

Thank You