GIT & GITHUB

What is GIT?

- Version Control System - Tool helps tod track changes in code
- GIT (Global Information Tracker) is a version control system
- GIT:
  - Popular
  - Free & open source
  - Fast & Scalable
- Installed and maintained on our local system

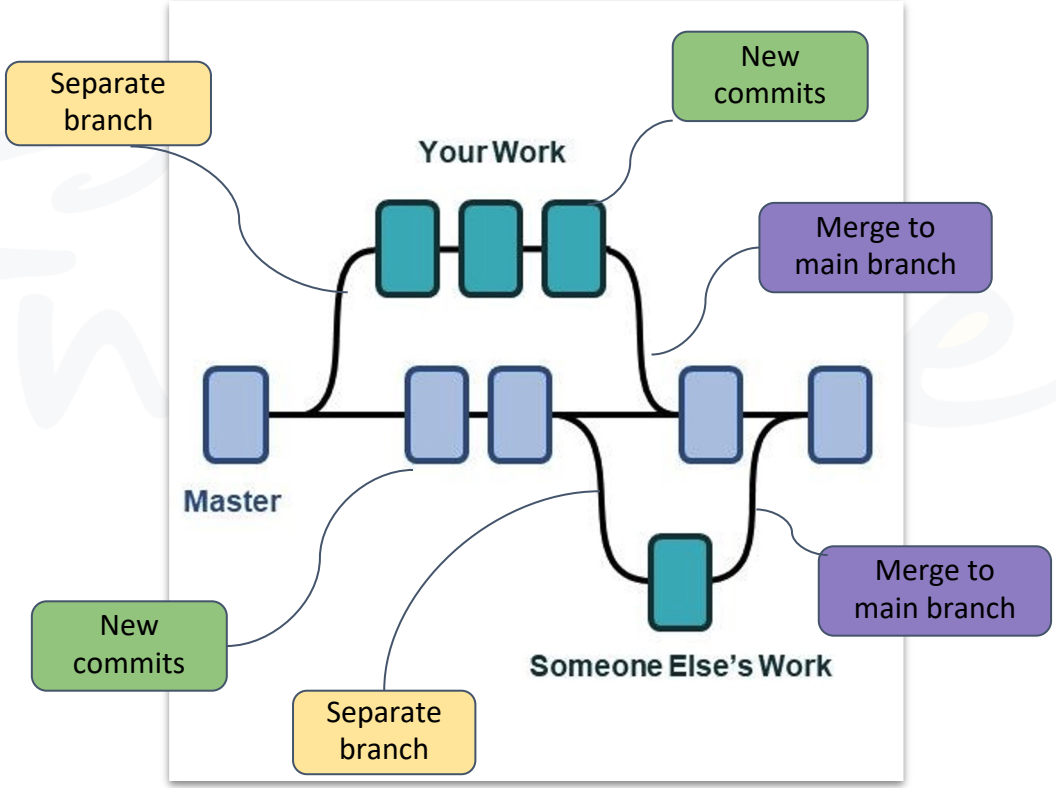
How GIT Helps?

- Track the history of changes
- Rollback to previous changes in couple of steps without any manual code changes
- Does not store a separate copy of each file in every commit, but keeps track of changes made in each commit. So does not use up more space
- GIT is not just for developers but for non-technical users as well by keeping track of their project files

What is Github?

- Web based hosting service for Git repositories built to run GIT in the cloud which allows developers to store and manage their code using GIT
- We can use GIT without GITHUB but cannot use GITHUB without GIT
- With Github, developers can create a single shared repository in their cloud
- Github owned by Microsoft

Overview


Separate branch
New commits
Your Work
Merge to main branch
Master
New commits
Separate branch
Someone Else's Work
Merge to main branch

Installing Git...

GIT

For Mac user

- MacOS or Linux Users will have Git installed by default
- Command to check:

git - - version

For Windows user

https://git-scm.com

Text Editor

VS Code

Created by Microsoft and have direct integrations with GitHub

https://code.visualstudio.com/

GIT Commands

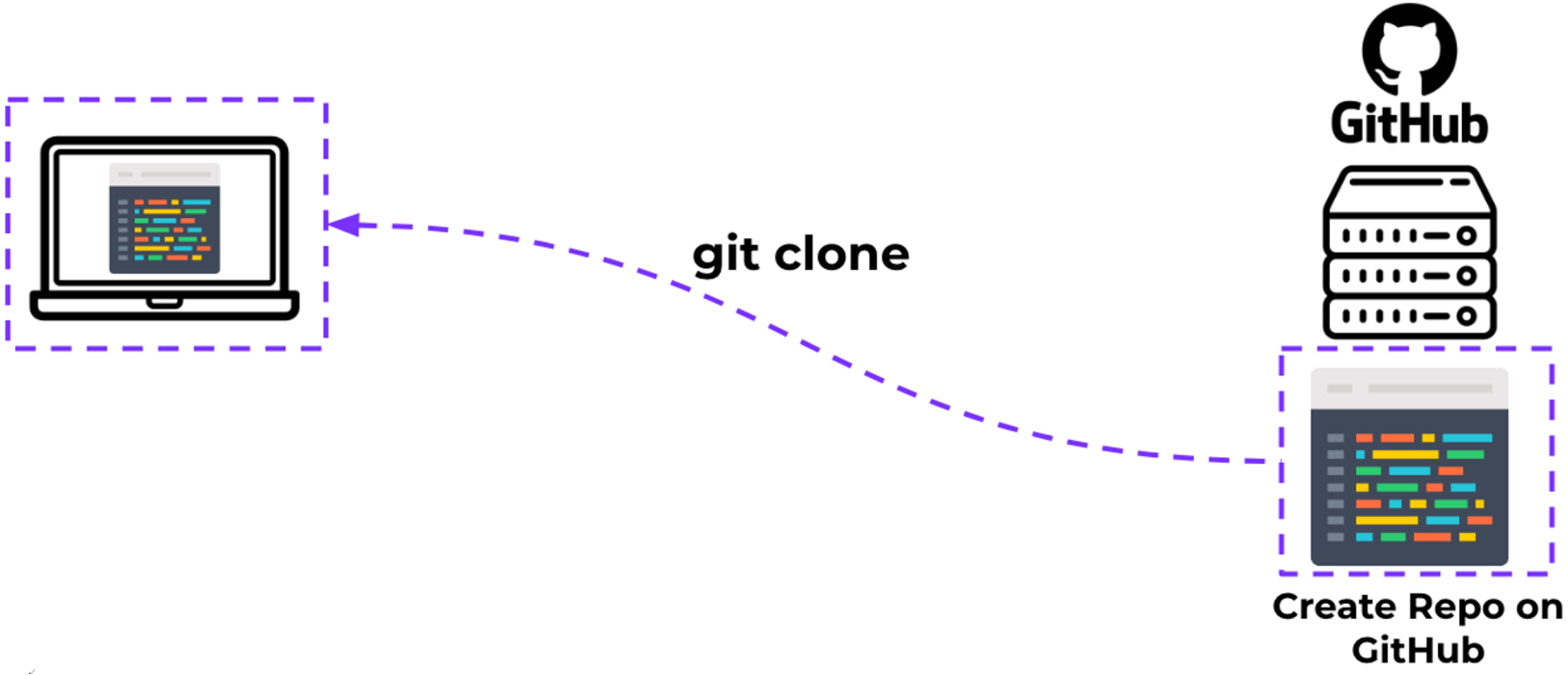
Git config

To perform an initial commit without the Git tool pestering for a *user.name* and *user.email* address command:

gitconfig -- global user.name “<>” **gitconfig –global user.email “<email>”**

- gitconfig – the extensionless system Git config file
- .gitconfig – the global Git config file has no name, but is instead just an extension
- config – the local Git configuration file is simply named *config,* and like the system Git config file, has no extension
- config.worktree – the Git worktree configuration file flaunts both a name and an extension

Git clone

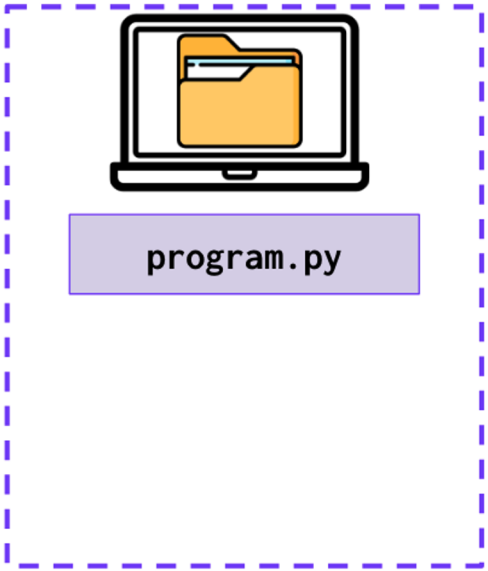


Use case story part-1

Working Directory

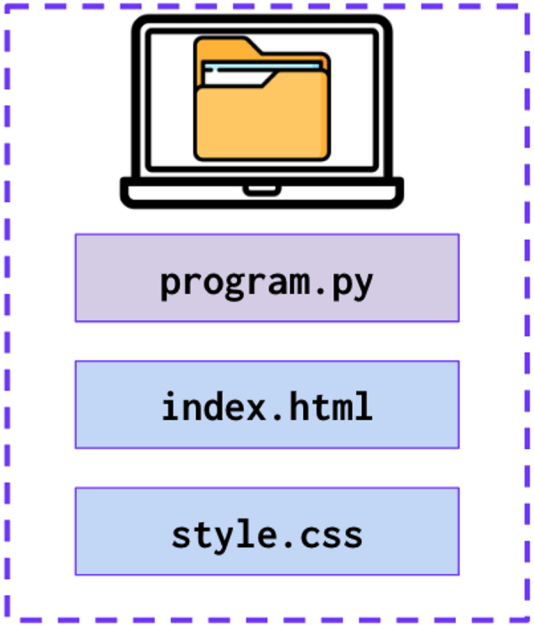
Use case story part-2

Working Directory



Use case story part-3

Working Directory



Use case story part-4

Working Directory

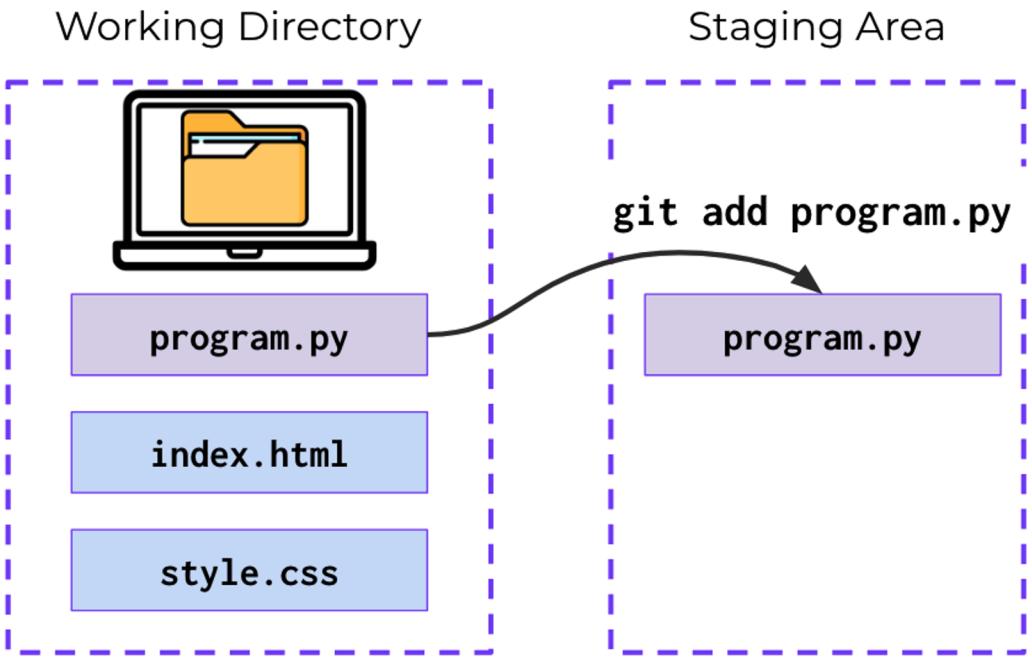
program.py

index.html

style.css

Staging Area

Use case story part-5



Use case story part-6

Working Directory

program.py

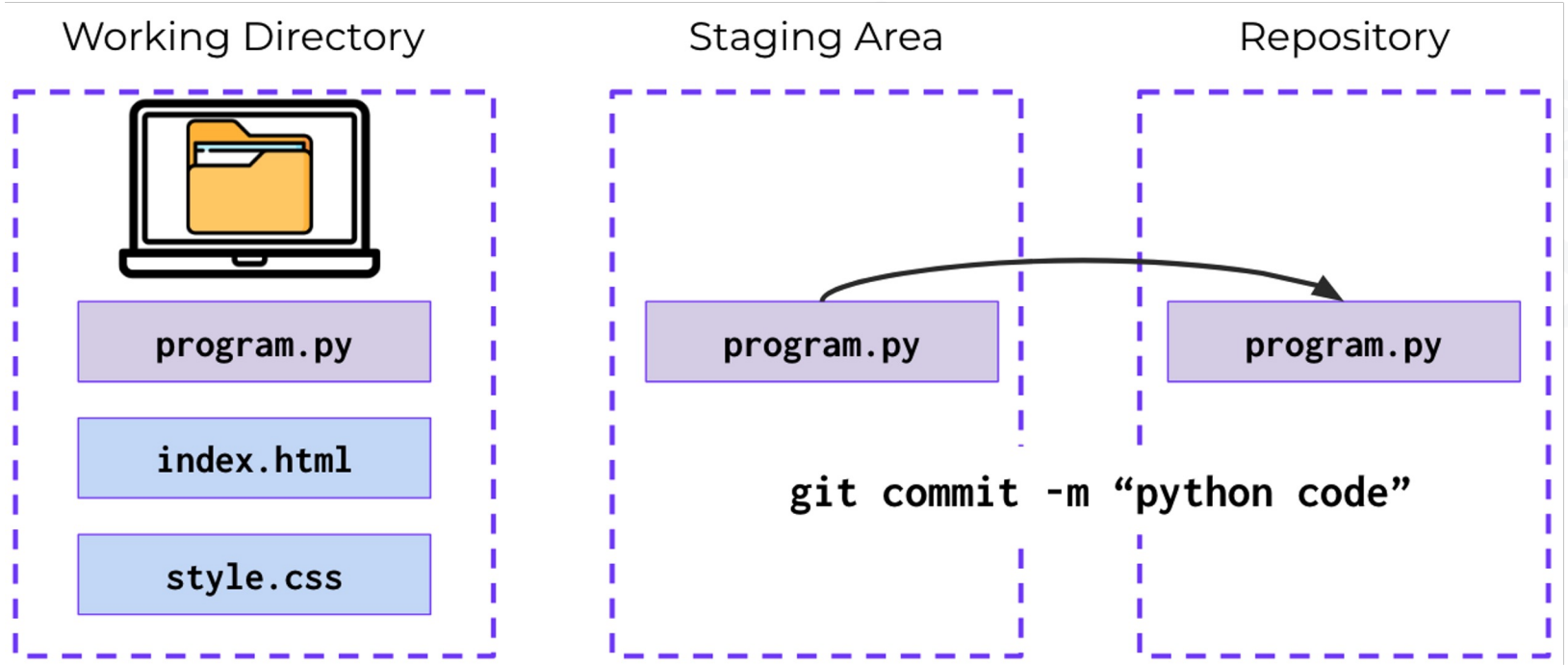
index.html

style.css

Staging Area

program.py

Use case story part-9



Use case story part-10

Working Directory

program.py

index.html

style.css

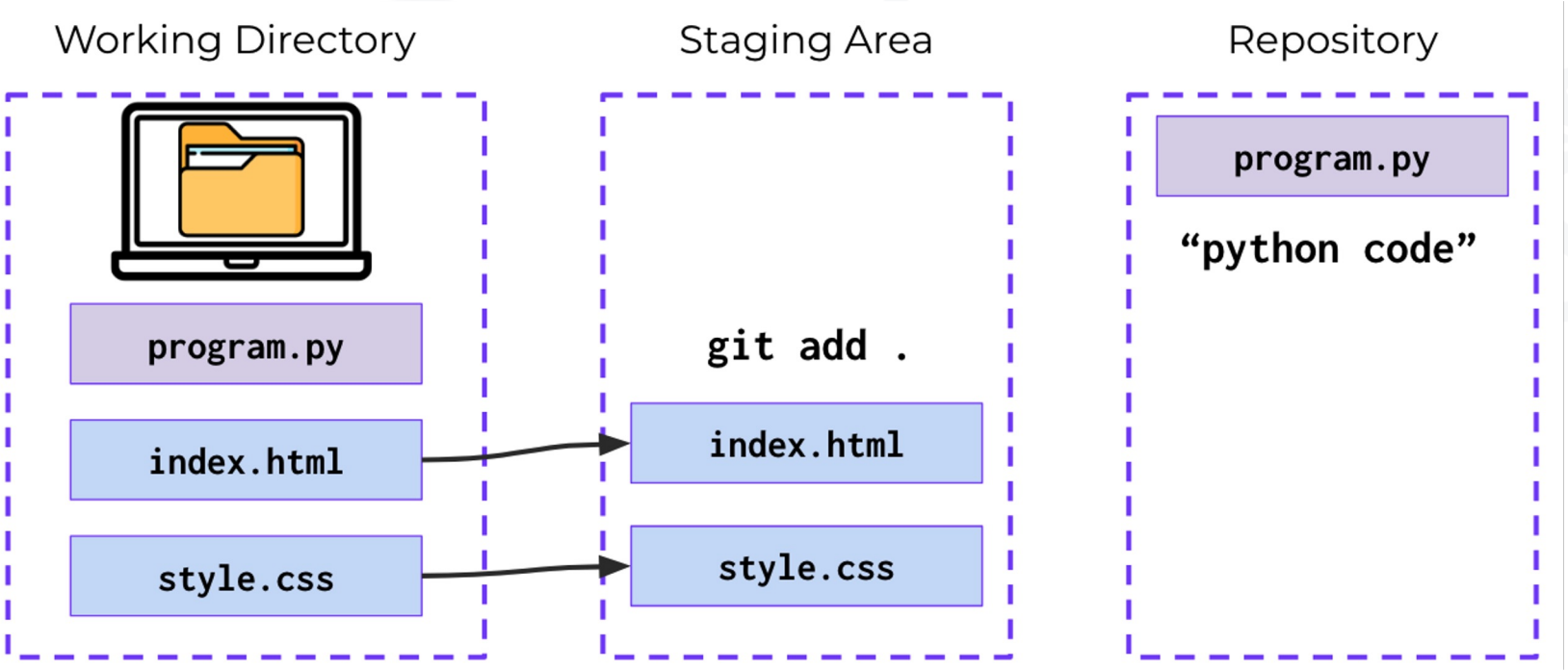
Staging Area

Repository

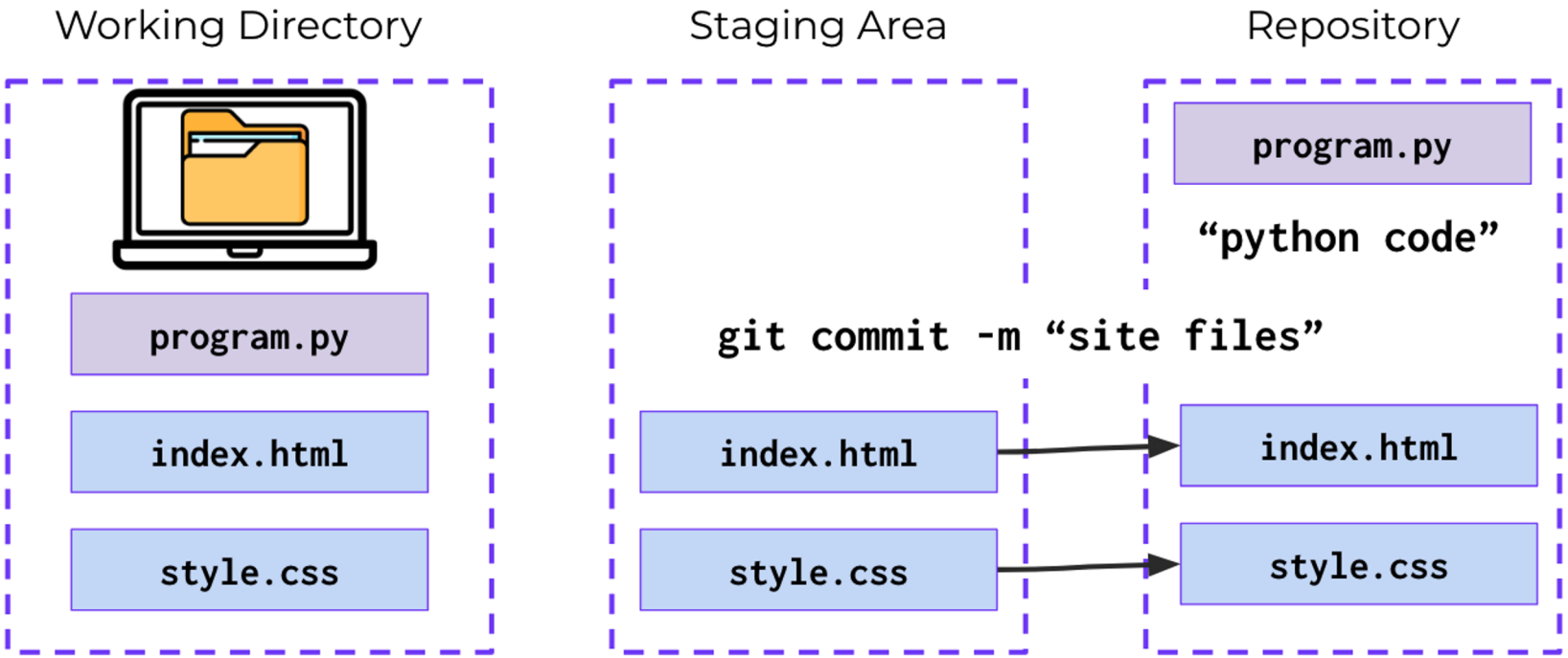
program.py

“python code”

Use case story part-11



Use case story part-12



Use case story part-13

Working Directory

program.py

index.html

style.css

Staging Area

Repository

program.py

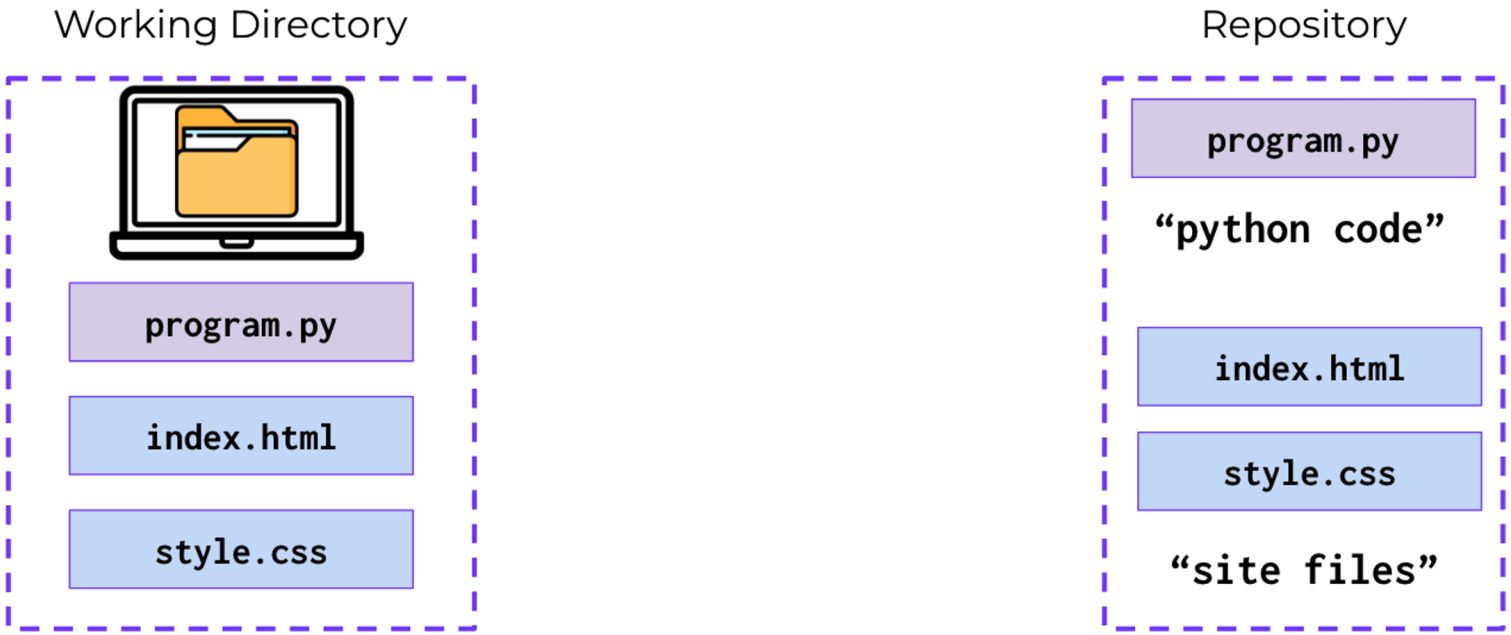
“python code”

index.html

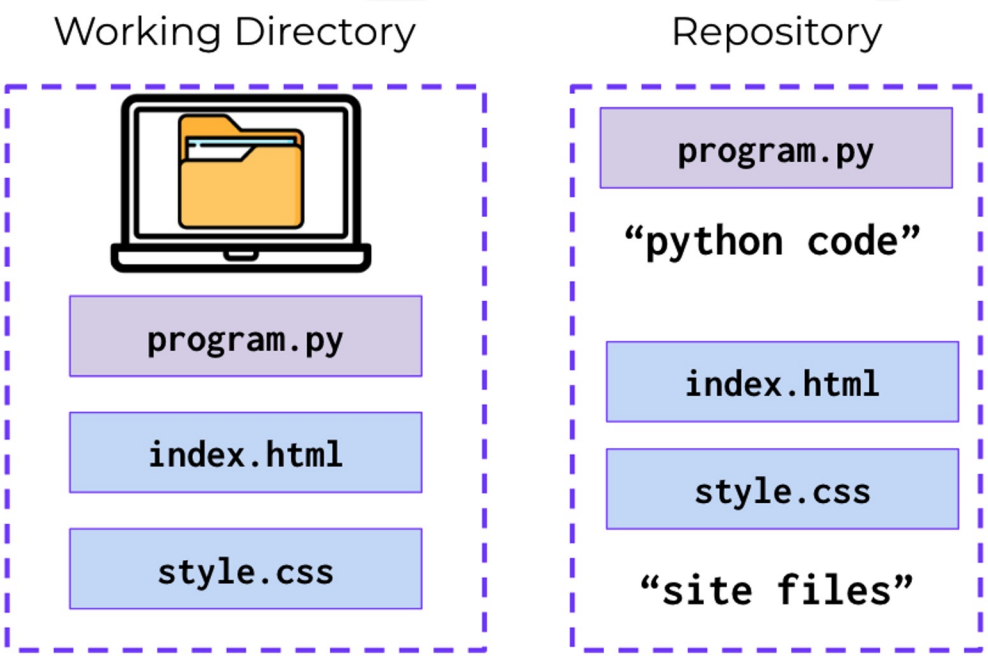
style.css

“site files”

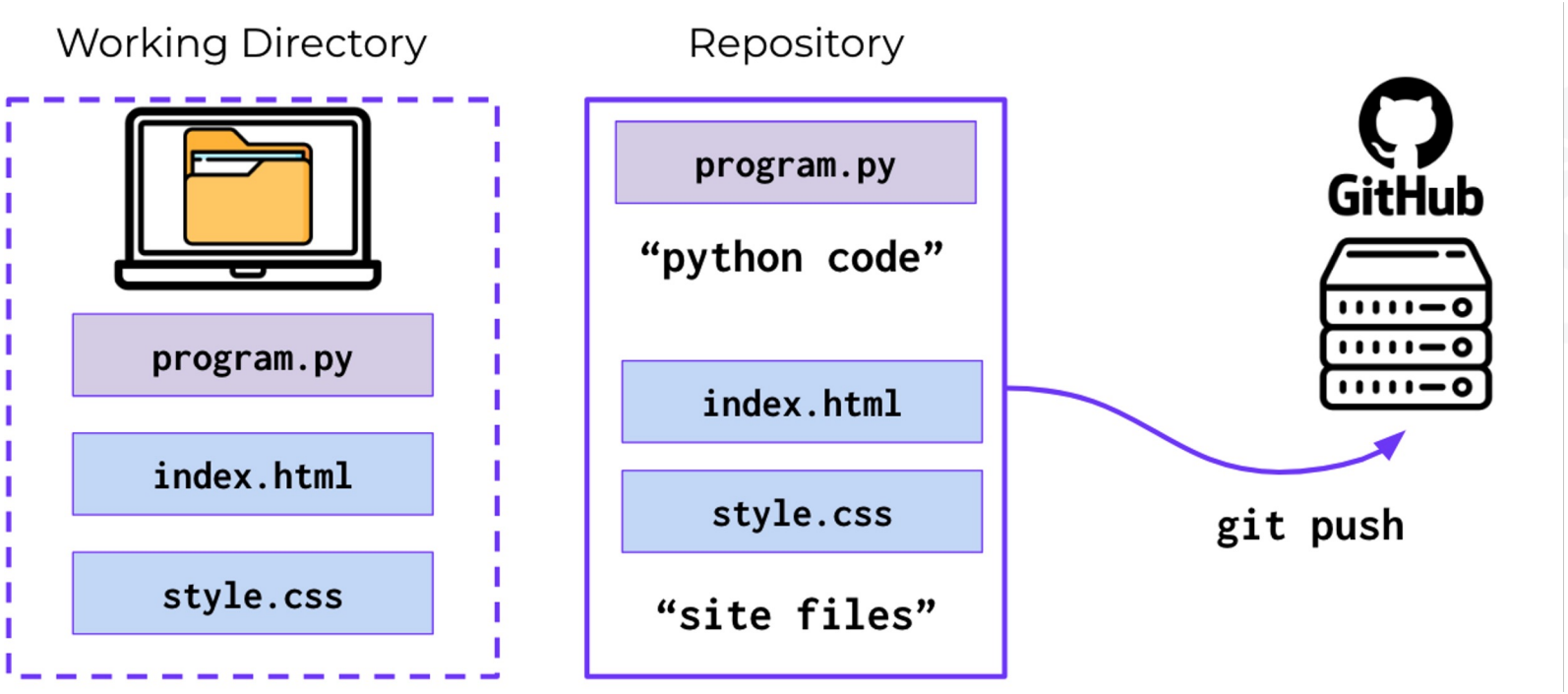
Use case story part-14



Use case story part-15



Use case story part-16



Use case story part-17

Working Directory

program.py

Repository

program.py

“python code”

index.html

style.css

“site files”

Use case story part-18

Working Directory

program.py

Repository

program.py

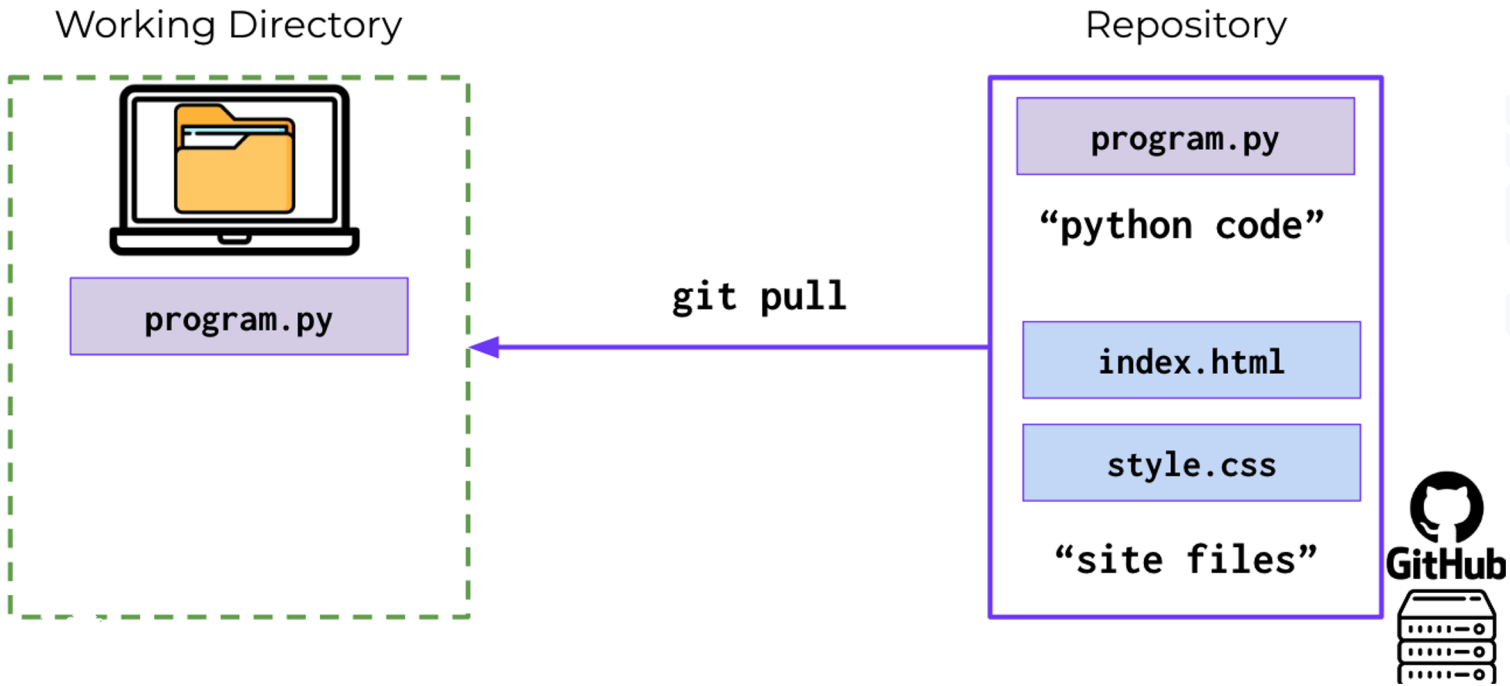
"python code"

index.html

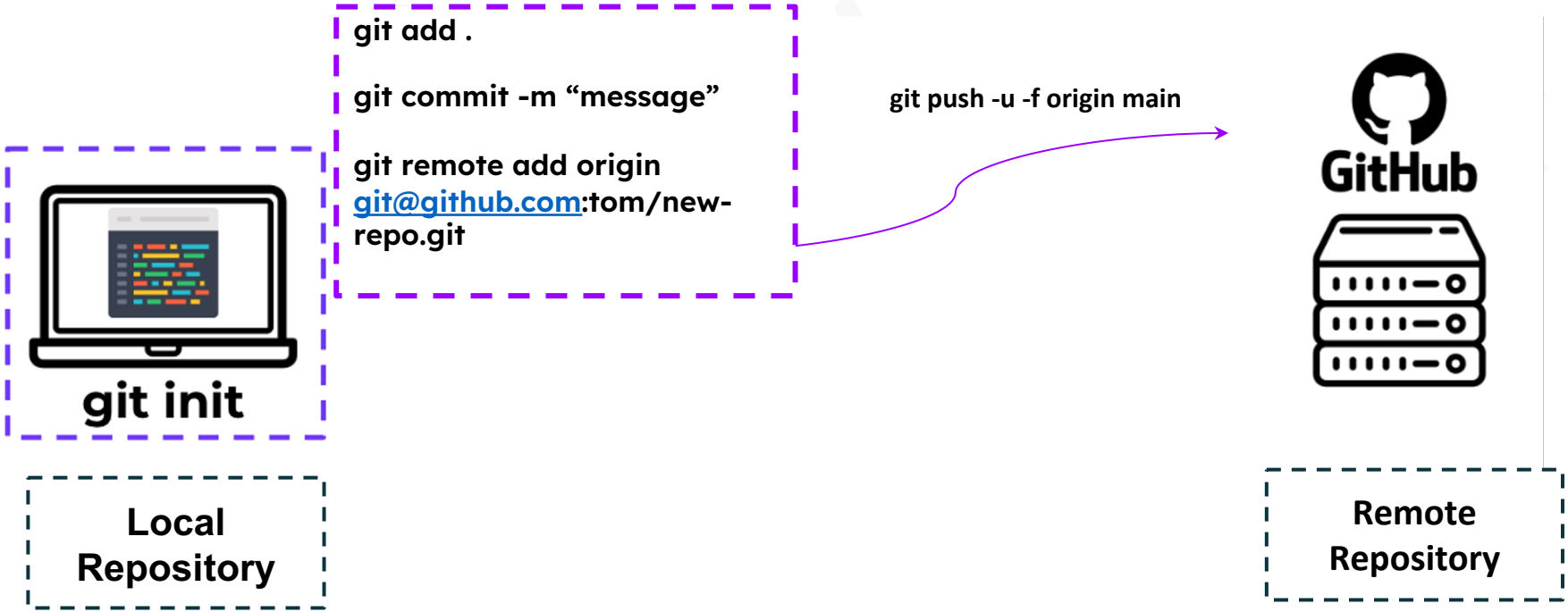
style.css

"site files"

Use case story part-16



Git init


git add .
git commit -m “message”
git remote add origin git@github.com:tom/new-repo.git
git push -u -f origin main
GitHub
git init
Local Repository
Remote Repository

Git init

- Initialise a local Git repository
- In the command **git remote add origin git@github.com:tom/new-repo.git**

Default name git gives to a remote server. We can have multiple remotes

Remote repo URL

- git push -u -f origin main

main branch

flag sets the remote origin as the upstream reference

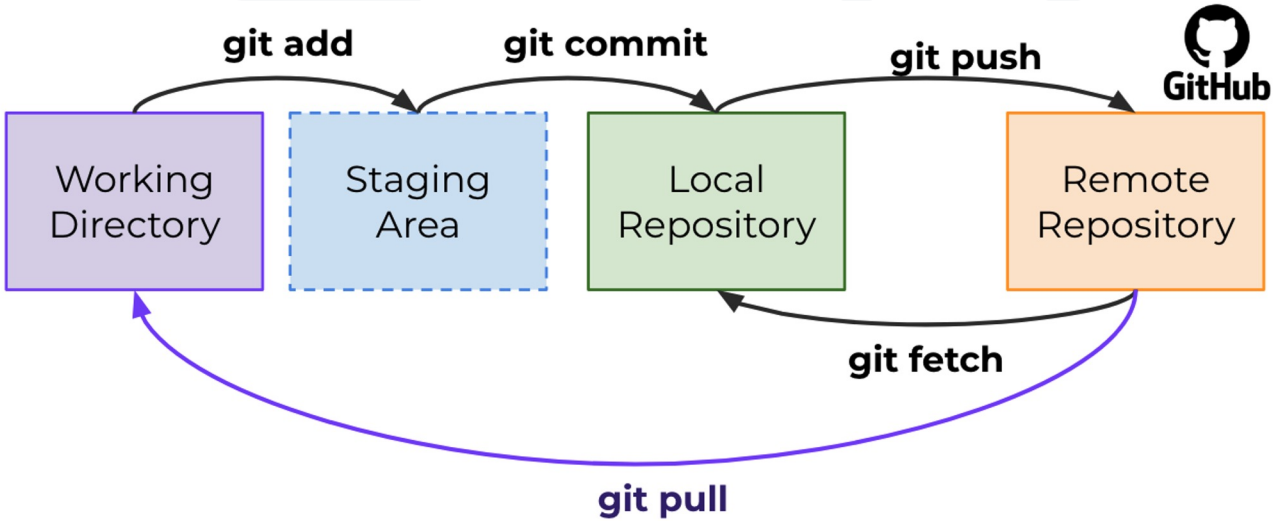
flag stands for force. Automatically overwrite everything in the remote directory

Remote server

Other commands

- git diff - Shows diff of changes made just before staging the changes
- git status - Shows the status of files in git at any point in time
- git pull - Pulls the changes from the remote repo to local
- git push - Pushes the changes from local repo to remote
- git checkout - create a new branch or switch between branches

Git fetch



git pull = git fetch + (git merge or git rebase)

git fetch used to keep local repo up to date but not to disturb working directory

To know history

- Two major ways to view the git history
  - git reflog:
    - check and read the history of everything that has happened to a repository
    - available history of the git HEAD ref. Keep tracks of deleted commits & branches as well
  - git log:
    - public record of the commit history of the repository
    - outputs from the current branch by default. deleted commits or branches are ignored

Recovery for mistakes

- Undoing **Uncommitted** changes
  - `git stash`: The git stash command will discard all your untracked files, as well as staged and unstaged modifications. However, Git will temporarily save them, in case you want to retrieve them later.
  - `git checkout --<file>`: This works similarly to git stash, except that it discards changes to a file permanently.
  - `git reset --hard`: This also discards all changes permanently.

Recovery for mistakes

- Undoing **Committed** changes
  - Run git reflog to see a history of all commits made.
  - Then note down the ID (SHA-1 hash) of the commit we want to go back to.
  - Now run git reset --hard <commit id>
- Undoing last commit
  - To amend last commit by running command git commit –amend
  - We need to stage our new changes with git add before running above command

git stash

- Git stores latest stash in ref/stash and name it stash@{0}
- Each older stash is stored and assigned an index number (ex: stash@{1}, stash@{2} etc..
- To know all the stashes, the command is git stash list
- Two ways to restore a Git stash:
  - git stash pop
  - git stash apply stash@{n}