CSE 470
Software Engineering
Git and Github

Imran Zahid
Lecturer
Computer Science and Engineering, BRAC University

BRAC UNIVERSITY
Inspiring Excellence

Git

What is Git?

- A distributed version control system
  - A Version Control System (VCS) is software that manages changes to files, especially source code, over time.
  - Distributed means that instead of having a single central repository (like with traditional VCS), every contributor has a full copy of the entire repository on their local machine, including the entire history of changes.
  - Allows multiple people to work on a project simultaneously without overwriting each other's work

What is Git?

- Created by Linus Torvalds in 2005
  - Developed to manage the Linux kernel development
  - Initially, a solution to the shortcomings of previous systems

Key Features of Git

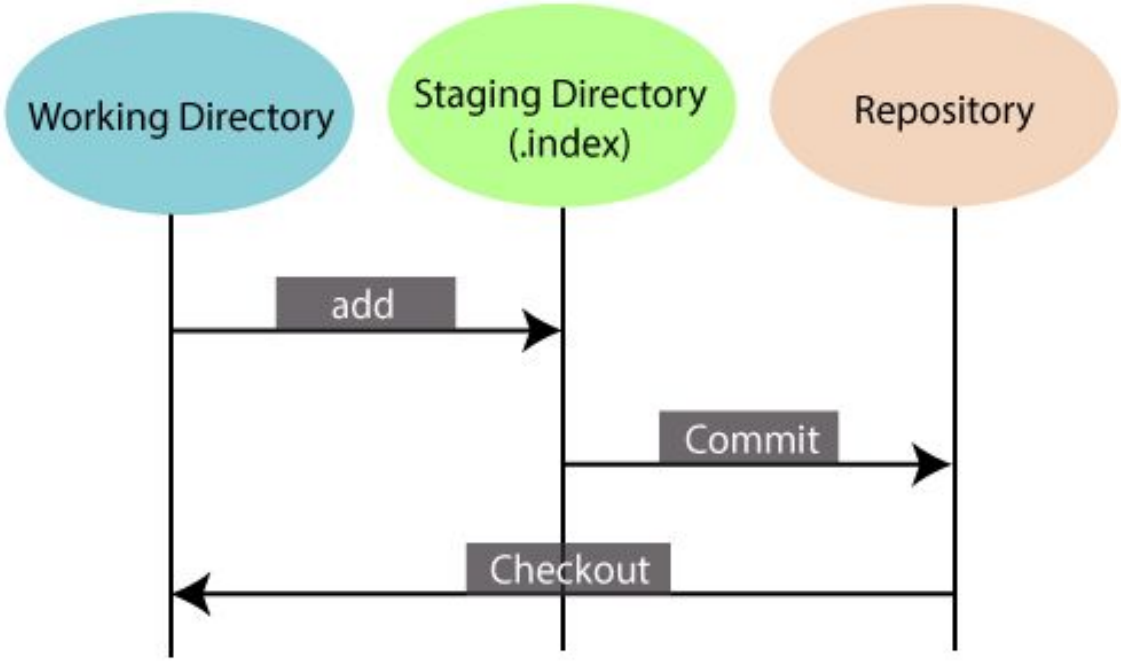
- Distributed
  - Each developer has a complete history of the project
- Fast Performance
  - Optimized for speed in managing changes
- Strong Support for Non-linear Development
  - Branching and merging capabilities

Using Git

- Repository
  - When you start a project with Git, you create a repository, which is like a folder that tracks all your changes.
- Commits
  - As you work on your project, you can save snapshots of your code, known as commits. Each commit records what changed and why. If you make a mistake, you can revert to a previous commit.
- Branches
  - Git also has a powerful branching system, letting you create separate lines of development within the same project. Once your changes are ready, you merge the branch back into the main project.

Basic Git Workflow

- Working Directory
  - Your local directory of files
- Staging Area (Index)
  - Files marked for the next commit
- Repository
  - Your project's complete history



Initial Git Setup

Add your user information to git. This does NOT have to match your github credentials, but it is better to do so.

- git config --global user.email “you@example.com”
- git config --global user.name “Your Name”

Initialize a Repository

- git init
  - Creates a new Git repository
  - Sets up initial structures
- git clone
  - Creates a copy of an existing repository

Basic Workflow - Version Control

1. Check the current status of working directory using “git status”
2. Tells you what files are untracked, if they were modified etc.
3. Add all files to the “stage” i.e. ready for commit but not committed yet using “git add --all”
4. Confirm it worked using “git status”
5. Commit all files in the staging area in a single commit with a particular commit message using “git commit -m “<<insert commit message here>>””
6. Confirm it worked using “git status”

Basic Workflow - Reverting Changes

- Undo Changes Before Adding to Staging (Working Directory)
  - If you modified a file but haven’t staged it yet, you can undo those changes:
  - git checkout -- <filename>

- Undo Staged Changes (Before Committing)
  - If you’ve added changes to the staging area (using git add), but you realize you need to undo that staging before committing:
  - git reset <filename>

Basic Workflow - Reverting Changes

- Undo a Commit (Keep Changes)
  - If you made a commit but realize you need to undo it while keeping the changes in your working directory:
  - git reset --soft HEAD~1

- Undo a Commit (Discard Changes)
  - If you want to undo a commit and completely discard its changes:
  - git reset --hard HEAD~1

Basic Workflow - Reverting Changes

- Viewing Commit History
  - To see the commit history and identify which commit you may want to revert to:
  - git log

- Revert a Specific Commit (Without Affecting Later Commits)
  - If you want to undo the effects of a specific commit in the history without affecting subsequent commits, you can use:
  - git revert <commit-hash>

Basic Workflow - Reverting Changes

- Permanently Move to a Specific Commit and Keep Changes (Mixed Reset)
  - This will reset your branch to a specific commit but keep the changes from later commits in your working directory.
  - git reset --mixed <commit-hash>

- Permanently Move to a Specific Commit and Discard Changes (Hard Reset)
  - If you want to reset the branch to a specific commit and discard all changes from later commits:
  - git reset --hard <commit-hash>

Basic Workflow - Branching

- Multiple developers shouldn’t work on the same branch at once because they might change the same file together leading to conflicts. Also if anything breaks in the main branch, the problems are harder to solve.
- For that reason, all changes MUST be made in a new branch, and after it passes quality checks, the new branch will be merged into the main branch.

Basic Workflow - Branching

- Create a new branch using
  - git branch <<insert branch name>>
- Switch to this new branch using
  - git checkout <<insert branch name>>
- Write code, and use the steps described in the basic workflow in order to commit the work done to this branch
- Once the work on this branch is completed, switch back to the main branch using
  - git checkout main

Basic Workflow - Branching

- Then merge the commits in the new branch into the main branch using
  - git merge <<insert branch name>>
- If all goes well, the changes done in the new branch will be merged into the main branch and you can then delete the unneeded branch using
  - git branch -d <<insert branch name>>
- However, sometimes conflicts can occur in the merging (i.e. two developers changed the same line). In that case we have to resolve conflicts (see next section)

Basic Workflow - Resolving Merge Conflicts

- If there are conflicts in the merge, git will show an error.
- In that case, open up the unmerged files, and see what is inside them

Basic Workflow - Resolving Merge Conflicts

Basic Workflow - Resolving Merge Conflicts

- Here, it says the contents of HEAD (remember head means the contents of last commit in the current branch) is print(“Hey”) and the contents of “feature/merge-testing” branch is print(“Hello”).
- We edit the contents of this file to remove the “<<<<<<”, “======” and “>>>>>>” lines and choose which of the two conflicting code sections we will choose to keep. Suppose we chose to keep print(“Hello”)

Basic Workflow - Resolving Merge Conflicts

- Then we add the files to the staging area again, and perform a new commit to commit the merge using
  - git commit -m <<Insert merge commit message>>
- Now the main branch contains all changes it previously made, along with the changes made in the new branch. You can verify this using
  - git log

Git Summary

- Create a git repository.
- Create a new branch for a particular task.
- Modify/add code
- Commit the changes with a descriptive message
- Merge the branch into main (resolve conflicts where necessary)

BRAC UNIVERSITY
Inspiring Excellence

Github (Remote Git Repository)

Remote Repository

- A remote repository is a version of your Git repository hosted on a server, such as GitHub, GitLab, or Bitbucket, that can be accessed and collaborated on by multiple users.
- It serves as a centralized location where team members can push their local changes and pull updates from others.

Github - New Repository

- Create your github account (https://github.com/)
- Create a new github repository
- I recommend not adding README or gitignore, if you already have a local repository with some files. If you are starting from scratch on github, use a different approach (see next section)
- You can follow the instructions on the empty repository to upload your existing repository, or create a new one.

Github - Pushing Local Files to the Remote

- When you come to the step where you push the local repository, you will be prompted for authentication.
- Once you have authenticated, you will successfully push your local files to your remote repository (GitHub)
- You can also push branches to the remote repository, by using “git push -u origin <<insert branch name>>”

Github - Merging Branches

- You can also push branches to the remote repository, by using “git push -u origin <<insert branch name>>”
- Once you have made changes to your branch, you can then perform the merging on GitHub itself. This is the preferred method for collaboration on GitHub. You work on your branch, you push it, and then you create a Pull Request.
- The owner of the repository or other admins can then review the pull request and choose to merge the pull request.

Naming Conventions

Git Commit Naming Conventions

<type>: <description>

Types

- feat : Commits, that adds or remove a new feature
- fix : Commits, that fixes a bug
- refactor : Commits, that rewrite/restructure your code, however does not change any API behavior
- chore : is for everything else (writing documentation, formatting, adding tests, cleaning useless code etc.)

Git Commit Naming Conventions

<type>: <description>

Description

- Think of the sentence “This commit will <<do this>>”
- In that case, the description should just include <<do this>>.
- Don't capitalize the first letter
- No dot (.) at the end

Git Commit Naming Conventions - Examples

Imagine each of these as the full sentence “This commit should …”.

- feat: add email notifications on new direct messages
- fix: add missing parameter to service call
- refactor: implement fibonacci number calculation as recursion
- chore: remove empty line

So the commit command would look like:

- git commit -m “feat: add email notifications on new direct messages”

Git Branch Naming Conventions

A git branch should start with a category. Pick one of these: feature, bugfix, or test.

- feature is for adding, refactoring or removing a feature
- bugfix is for fixing a bug
- test is for experimenting outside of an issue/feature

There should be a "/" followed by a description which sums up the purpose of this specific branch. This description should be short and "kebab-cased".

Examples

- git branch feature/create-new-button-component
- git branch bugfix/button-overlap-form-on-mobile

BRAC UNIVERSITY
Inspiring Excellence

Thank you