Version Control

Prepared by :
Sohaib, Lecturer, BUET CSE

01 Introduction to version control

- What is version control ?
- How version control helps in collaboration?
- Why use git ?

Version Control

Version control is a system that tracks and manages *changes* to source code, protecting it from errors and enabling *collaboration* among developers. It ensures that teams can work *concurrently* and efficiently while maintaining a complete *history* of every change.

Version Control

- *Tracking* individual changes from each contributor.
- Allowing *branching* for independent streams of work.
- Preventing *conflicts* between concurrent work.
- Facilitating *merging* of changes to maintain project integrity.
- Providing *traceability* for each change, linked to project management tools.

Why Git ?

The most widely used modern version control system in the world today is Git. Git is a mature, actively maintained open source project originally developed in 2005 by Linus Torvalds, the famous creator of the Linux operating system kernel.

- Supports flexible workflows
- Works on any platform
- Popular

Git is an example of a DVCS (Distributed Version Control System)

Installing Git

- Download Git for Windows (*https://gitforwindows.org/*)
- Open Git Bash
  - `$ git config --global user.name "JohnDoe"`
  - `$ git config --global user.email jdoe@gmail.com`

02 Git Basics

- How to create a git repository?
- How to save changes ?
- How to check differences ?
- How to undo changes ?

Git Init

The *git init* command creates a new Git repository. It can be used to convert an existing, unversioned project to a Git repository or initialize a new, empty repository.

Most other Git commands are not available outside of an initialized repository, so this is usually the first command you'll run in a new project

Git Init

Executing git init creates a .git subdirectory in the current working directory, which contains all of the necessary Git metadata for the new repository.

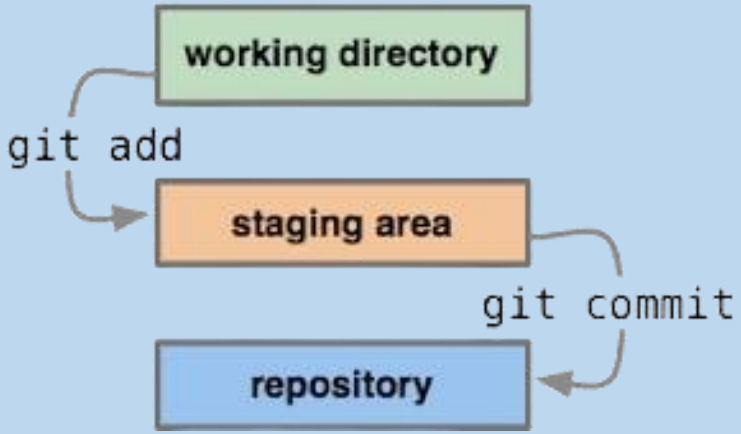
Saving Changes

The commands: git add, git status, and git commit are all used in combination to save a snapshot of a Git project's current state.

Saving Changes

The *git add* command stages changes from the working directory for the next commit, signaling Git that we want to include updates. However, changes aren't recorded until we run *git commit*. We can use *git status* to check the state of the working directory and staging area.

Saving Changes



git

Git Ignore

Git sees every file in your working copy as one of three things:

- Tracked
- Untracked
- Ignored

Amending Commits

Sometimes we want to make changes to our last commit.

- `git commit --amend`

It is commonly used when we want to *edit* the commit message or include *additional changes* that were left out by mistake. Rather than creating a new commit, this command replaces the previous one, combining the changes together.

03 Collaboration

- How to upload your repository to the web?
- What is the difference between git and github?
- How multiple users can work simultaneously?

git

Adding Remote Origin

The git remote command lets you create, view, and delete connections to other repositories

```
git remote add <name> <url>
```

```
git remote rm <name>
```

Adding Remote Origin

Invoking git remote with the -v option will print the list of bookmarked repository names and additionally, the corresponding repository URL. The -v option stands for "verbose".

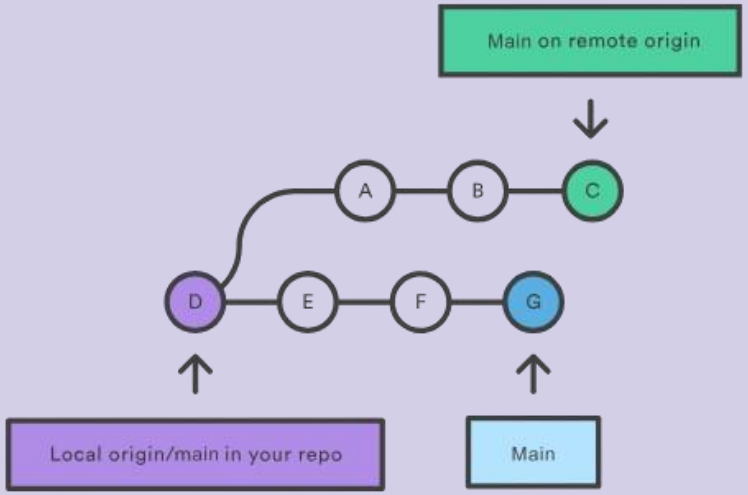
Git Push

The git push command is used to write to a remote repository.

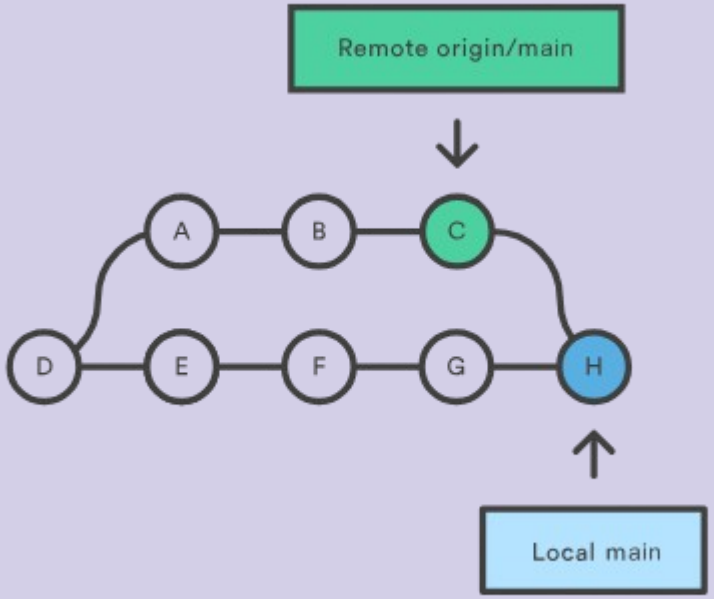
```
git push <remote-name> <branch-name>
```

Git Pull

The git pull command is used to fetch and download content from a remote repository and immediately update the local repository to match that content



Git Pull



The pull process will then create a new local merge commit containing the content of the new diverged remote commits. In the above diagram, we can see the new commit H.

Git Fetch

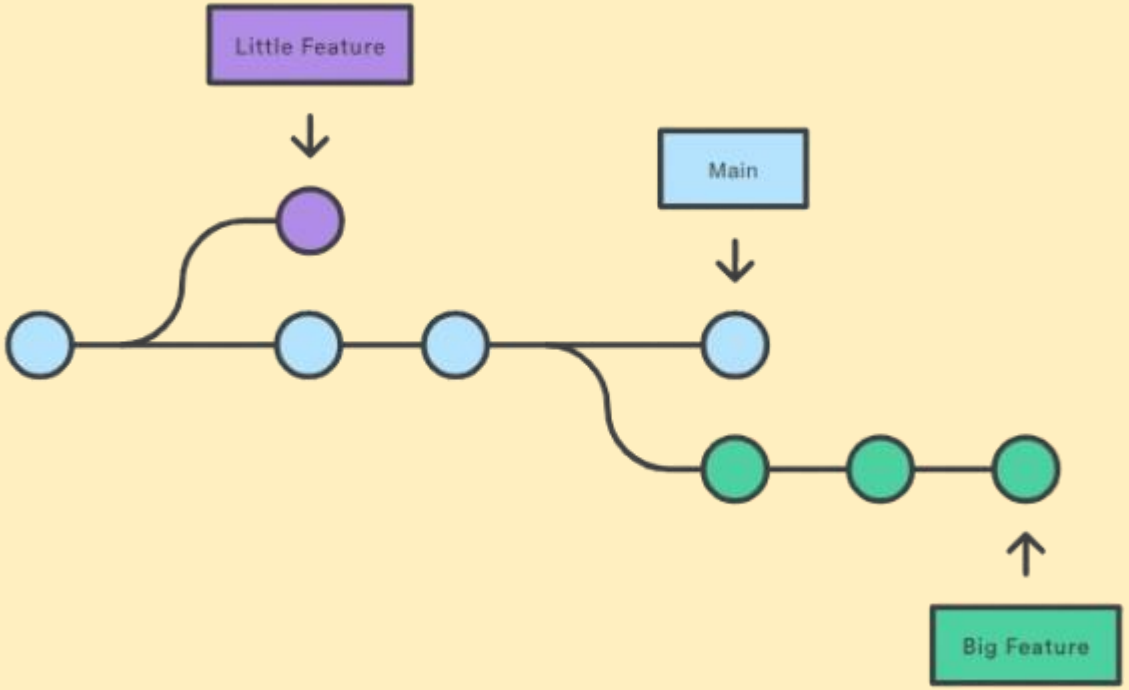
The git fetch command downloads commits, files, and refs from a remote repository into your local repo. Git isolates fetched content from existing local content, it has absolutely no effect on your local development work. git pull is the more aggressive alternative.

04 Branching

- How multiple users can work simultaneously?
- What are merge conflicts?
- How do we resolve conflicts?

git

Git Branches



Git Branches

A branch represents an independent line of development. Branches serve as an abstraction for the edit/stage/commit process. The git branch command lets you create, list, rename, and delete branches. It doesn't let you switch between branches or put a forked history back together again.

Git Branches

- `git branch`
- `git branch <branch>`
- `git branch -d <branch>`
- `git branch -D <branch>`

Git Branches

- `git push origin main`

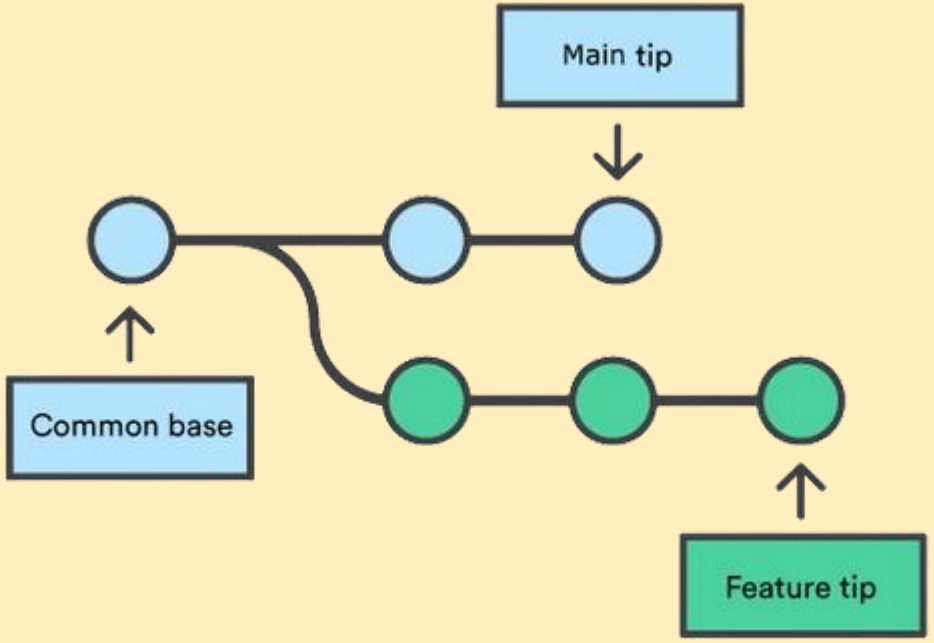
This will push the main branch into the remote counterpart.

Git Branches

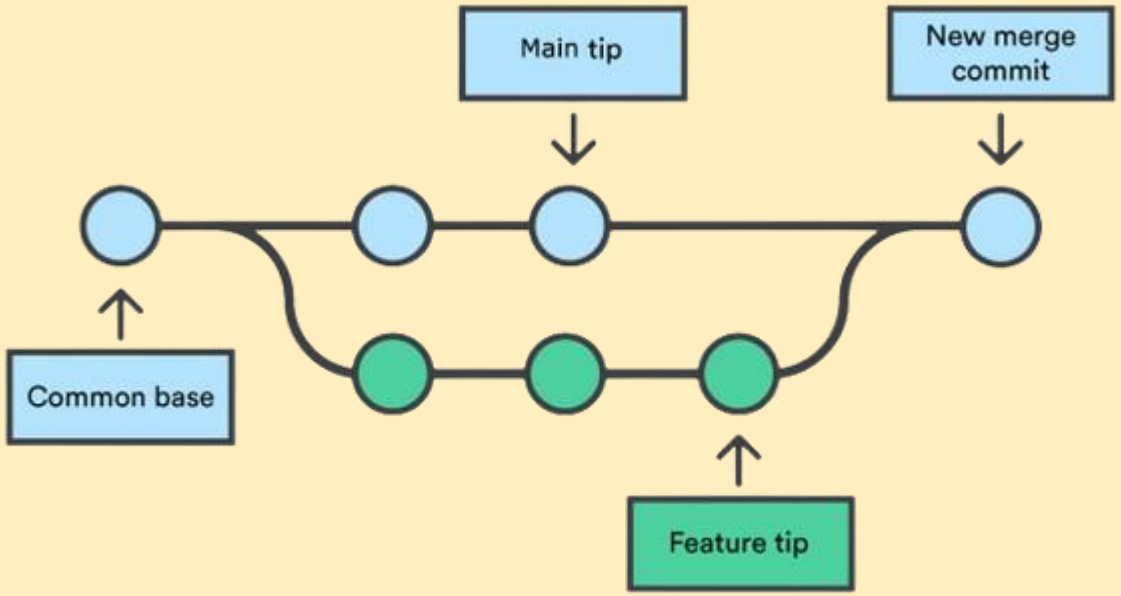
- `git checkout -b <branch>`

Create and check out a new branch named <branch>. Drop the -b flag to checkout an existing branch

Git Merge



Git Merge



Conflicts

If the two branches you're trying to merge both changed the same part of the same file, Git won't be able to figure out which version to use. When such a situation occurs, it stops right before the merge commit so that you can resolve the conflicts manually.

Conflicts

Generally the content before the ======= marker is the receiving branch and the part after is the merging branch.

```
here is some content not affected by the conflict
<<<<<<< main
this is conflicted text from main
=======
this is conflicted text from feature branch
>>>>>>> feature branch;
```

Conflicts

Once you've identified conflicting sections, you can go in and fix up the merge to your liking. When you're ready to finish the merge, all you have to do is run git add on the conflicted file(s) to tell Git they're resolved. Then, you run a normal git commit to generate the merge commit

05 Undoing Changes

- Git clean
- Git revert
- Git reset

git

Git Clean

The git clean command is a convenience method for deleting *untracked files* in a repo's working directory. Untracked files are those that are in the repo's directory but have not yet been added to the repo's index with git add

Git Clean

- `git clean -n` `# shows which files are going to be removed`
- `git clean -f` `# operates on the current dir untracked files`
- `git clean -di` `# interactive mode`

Git Revert

The *git revert* command simply creates a new commit that is the opposite of an existing commit.

Git Reset

```
git reset --hard <SOME-COMMIT>
```

- Makes current branch back to point at <SOME-COMMIT>
- Makes the files in your working tree and the index ("staging area") the same as the versions committed in <SOME-COMMIT>

Git Diff

The *git diff* is a multi-use Git command that when executed runs a diff function on Git data sources. These data sources can be *commits*, *branches*, *files* and more. The git diff command is often used along with git status and git log to analyze the current state of a Git repo.

Git Diff

By default git diff will show you any uncommitted changes since the last commit. But we can compare between two different commits by providing the checksum of the commits. We can also consider differences for only 1 file.

Git Stash

The *git stash* command takes your uncommitted changes (both staged and unstaged), saves them away for later use, and then reverts them from your working copy

Note that the stash is local to your Git repository; stashes are not transferred to the server when you push.

Git Stash

We can reapply previously stashed changes with git stash pop.

Alternatively, you can reapply the changes to your working copy and keep them in your stash with git stash apply.

Git Stash

You can run *git stash* several times to create multiple stashes, and then use *git stash list* to view them

To provide a bit more context, it's good practice to annotate your stashes with a description, using *git stash save "message“*

You can view a summary of a stash with *git stash show*

Git Ignore

Git sees every file in your working copy as one of three things:

- Tracked
- Untracked
- Ignored

Git Ignore

The .gitignore file normally stays at the top level directory of your repository. It uses globbing patterns to match against file names

- Logs (Any file or directories with this name)
- *.log (Asterisk acts as wildcard)
- !important.log (exclamation acts as negation)

Git Blame

The *git blame* command is a versatile troubleshooting utility that has extensive usage options. The high-level function of git blame is the display of author metadata attached to specific committed lines in a file

Thank You