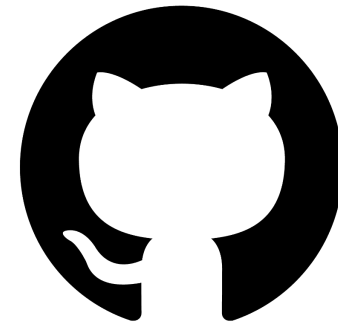




git



GitHub

What is Git?

Free and open source version control system

What is Version Control system?

- A system that keeps track of our files or projects.
- It allows you to revert selected files to a previous state, **revert** the entire project to a previous state, **compare changes** over time, see who last modified something so that we can know what might be causing a problem, or **what** is the issue, **who** made it, and **when** with the details.

Alice's Code



v1.0.0

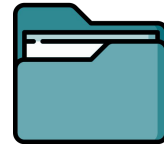


v1.0.1



v1.0.2

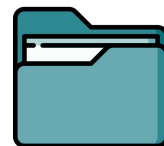
Bob's code



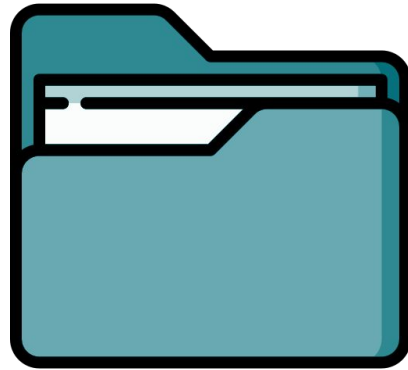
v1.0.0



v1.0.1



v1.0.2



v1.0.2

2 types of VCS

Centralized



```
graph TD; A[2 types of VCS] --> B((Centralized)); A --> C((Distributed));
```

The diagram illustrates the two types of Version Control Systems (VCS). At the top, a pink rounded rectangle contains the text "2 types of VCS". Below this, two light blue circles with dark blue outlines are positioned side-by-side. The left circle is labeled "Centralized" and the right circle is labeled "Distributed".

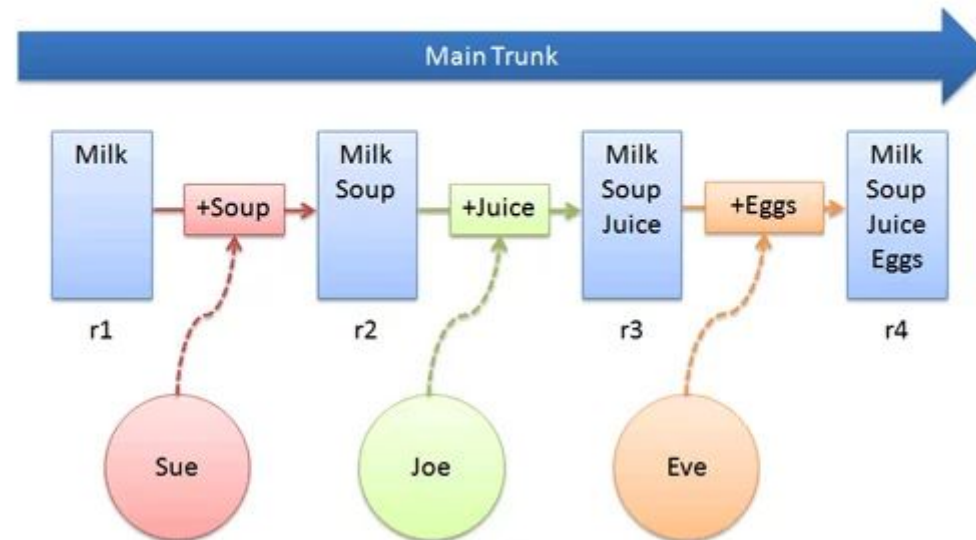
Distributed

Centralized version control

Helps you backup, track and synchronize files.

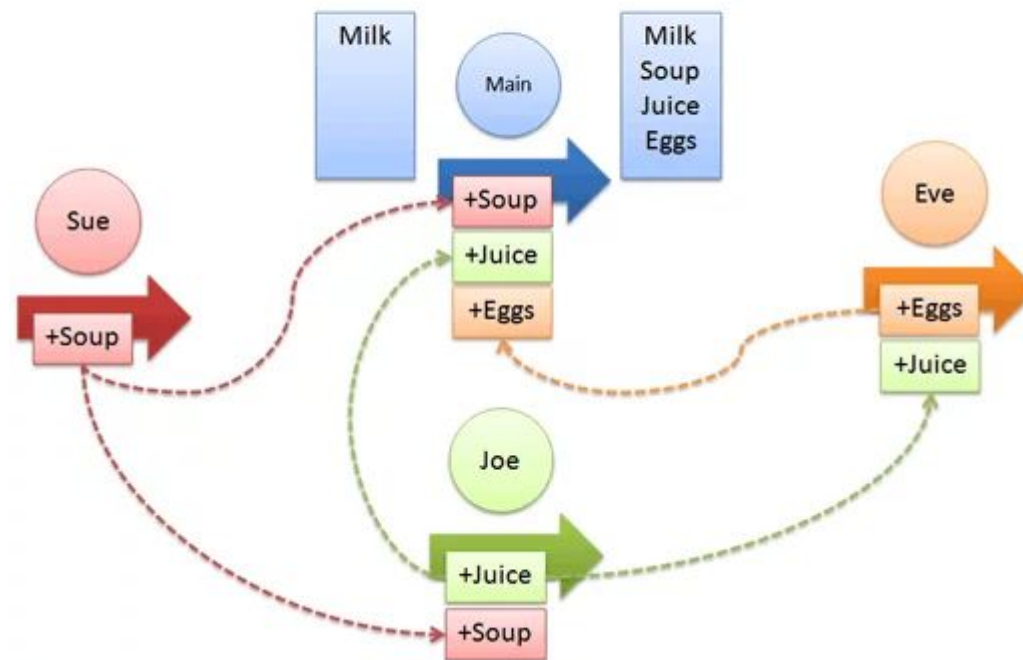
Centralized VCS

Eg: Subversion &
Team foundation
server



Distributed Version Control Systems

Distributed VCS



Eg: Git & Mercurial

Why Git?

- Free
- Open source
- Scalable
- Super Fast
- Cheap branching and merging

What is GitHub?

GitHub is a **web-based hosting service** for git repositories.

You can use git without Github, but you cannot use GitHub without Git.

Git	GitHub
Used for Version Control	Used for hosting Git repositories
Installed locally on computer	Cloud based
Tracks changes made to a file	Provides a web interface to view file changes

Local Repository

Every VCS tool provides a private workplace as a working copy. Developers make changes in their private workplace and after commit, these changes become a part of the repository. Git takes it one step further by providing them a private copy of the whole repository. Users can perform many operations with this repository such as add file, remove file, rename file, move file, commit changes, and many more.

Working Directory and Staging Area or Index: An intermediate area where commits can be formatted and reviewed before completing the commit.

push: send a change to another repository (may require permission)

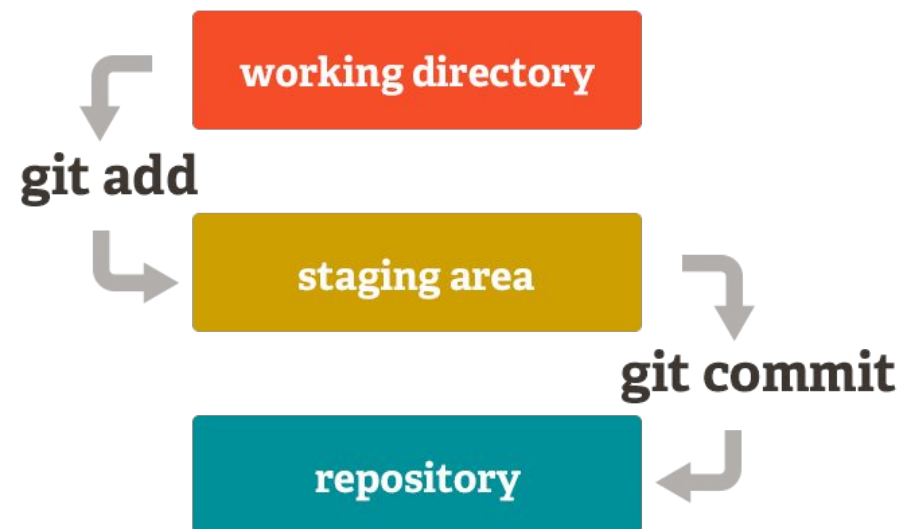
pull: grab a change from a repository

Basic workflow of Git.

Step 1 – You modify a file from the working directory.

Step 2 – You add these files to the staging area.

Step 3 – You perform commit operation that moves the files from the staging area. After push operation, it stores the changes permanently to the Git repository.



Source git-scm.com

Blobs

Blob stands for **B**inary **L**arge **O**bject. Each version of a file is represented by blob. A blob holds the file data but doesn't contain any metadata about the file. It is a binary file, and in Git database, it is named as SHA1 hash of that file. In Git, files are not addressed by names. Everything is content-addressed.

Trees

Tree is an object, which represents a directory. It holds blobs as well as other sub-directories. A tree is a binary file that stores references to blobs and trees which are also named as SHA1 hash of the tree object.

Commits

- Commit holds the current state of the repository. A commit is also named by SHA1 hash.
- Commit object = a node of the linked list.
- Every commit object has a pointer to the parent commit object.
- From a given commit, you can traverse back by looking at the parent pointer to view the history of the commit.
- If a commit has multiple parent commits, then that particular commit has been created by merging two branches.

Git commands

Clone: Bring a repository hosted somewhere like Github into a folder or your local machine

Add: Track your files and changes in Git

Commit: Save your files in git

Push: Upload your commits to a git repo, like GitHub

Pull: Download changes from a remote repository to your local repository.



Already have an account? [Sign in](#) →

Welcome to GitHub!
Let's begin the adventure

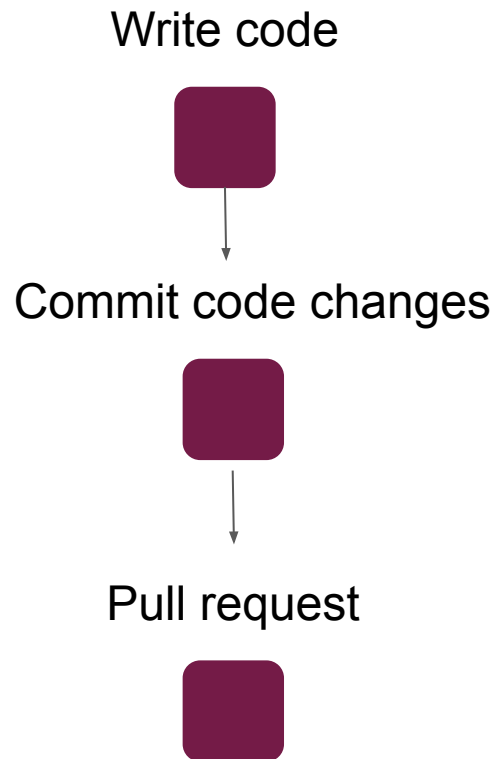
Enter your email*



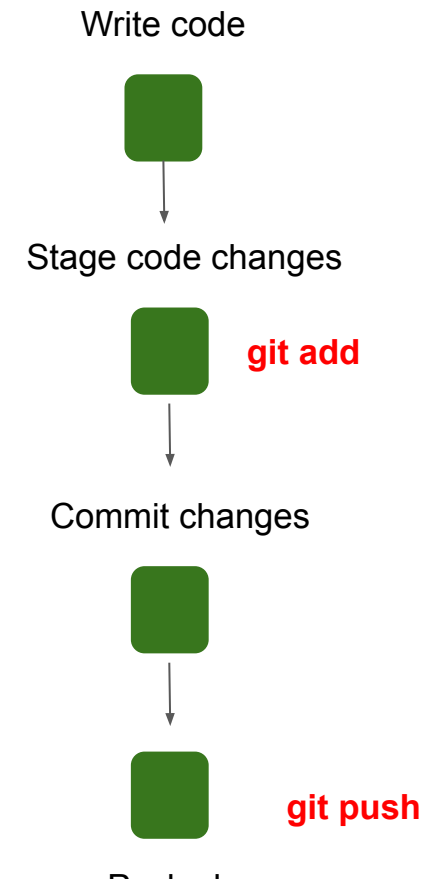
Continue

By creating an account, you agree to the [Terms of Service](#). For more information about GitHub's privacy practices, see the [GitHub Privacy Statement](#). We'll occasionally send you account-related emails.

GitHub workflow

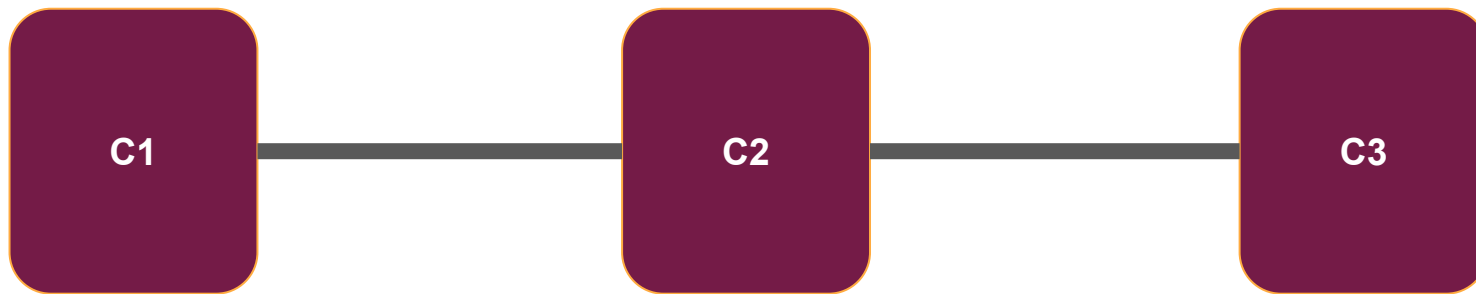


Local Git Workflow

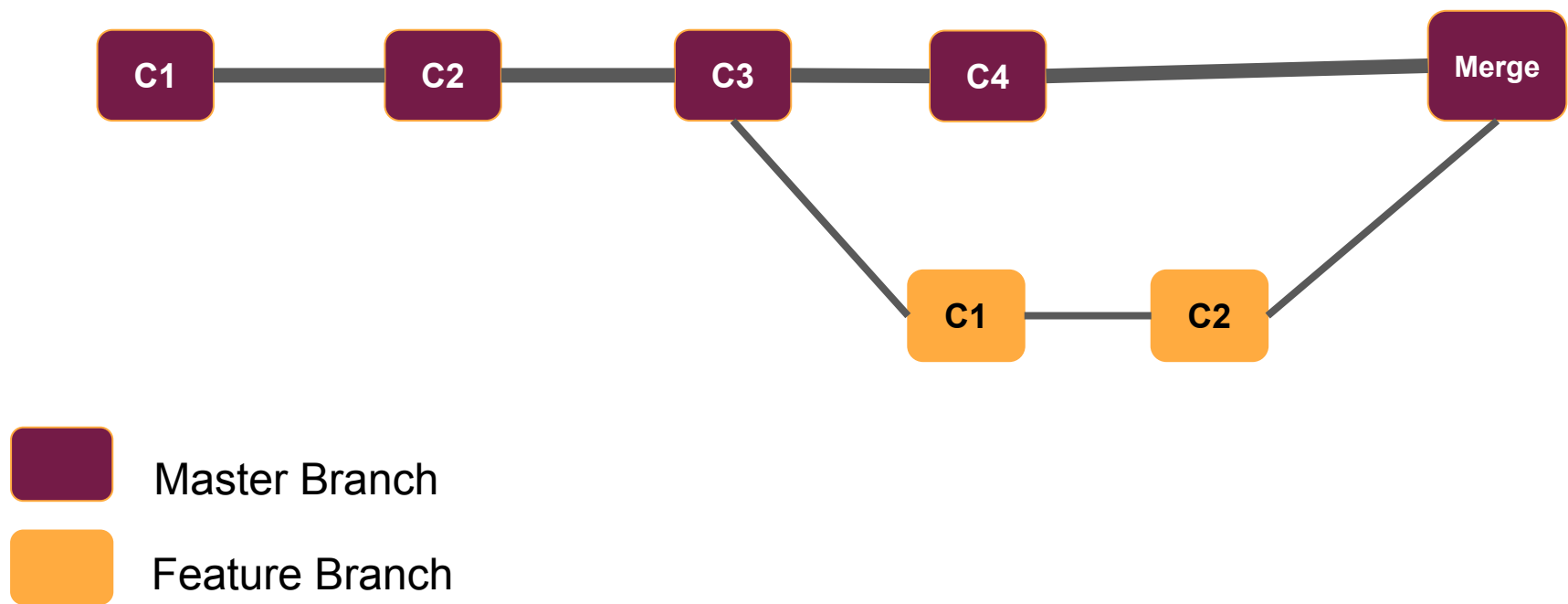


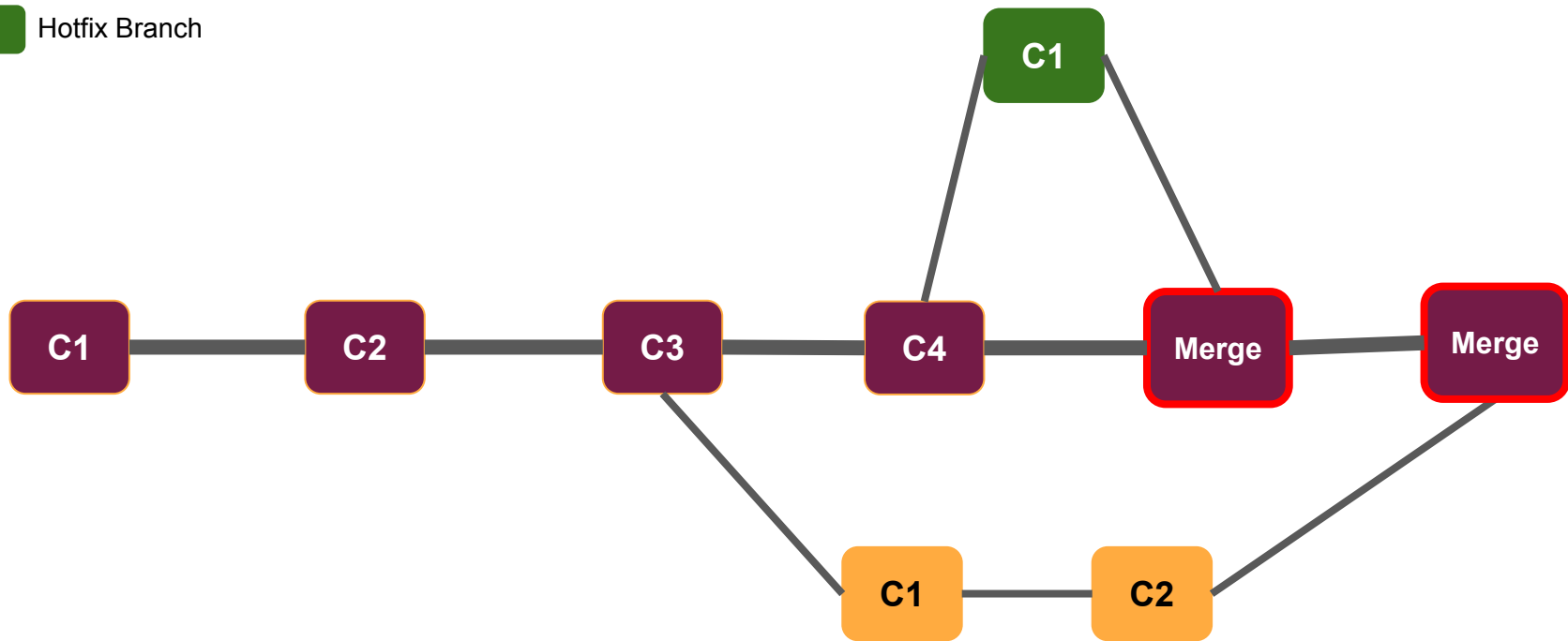
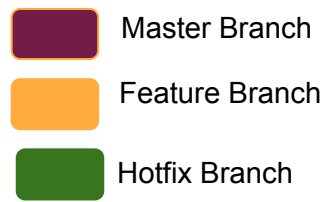
Git Branching

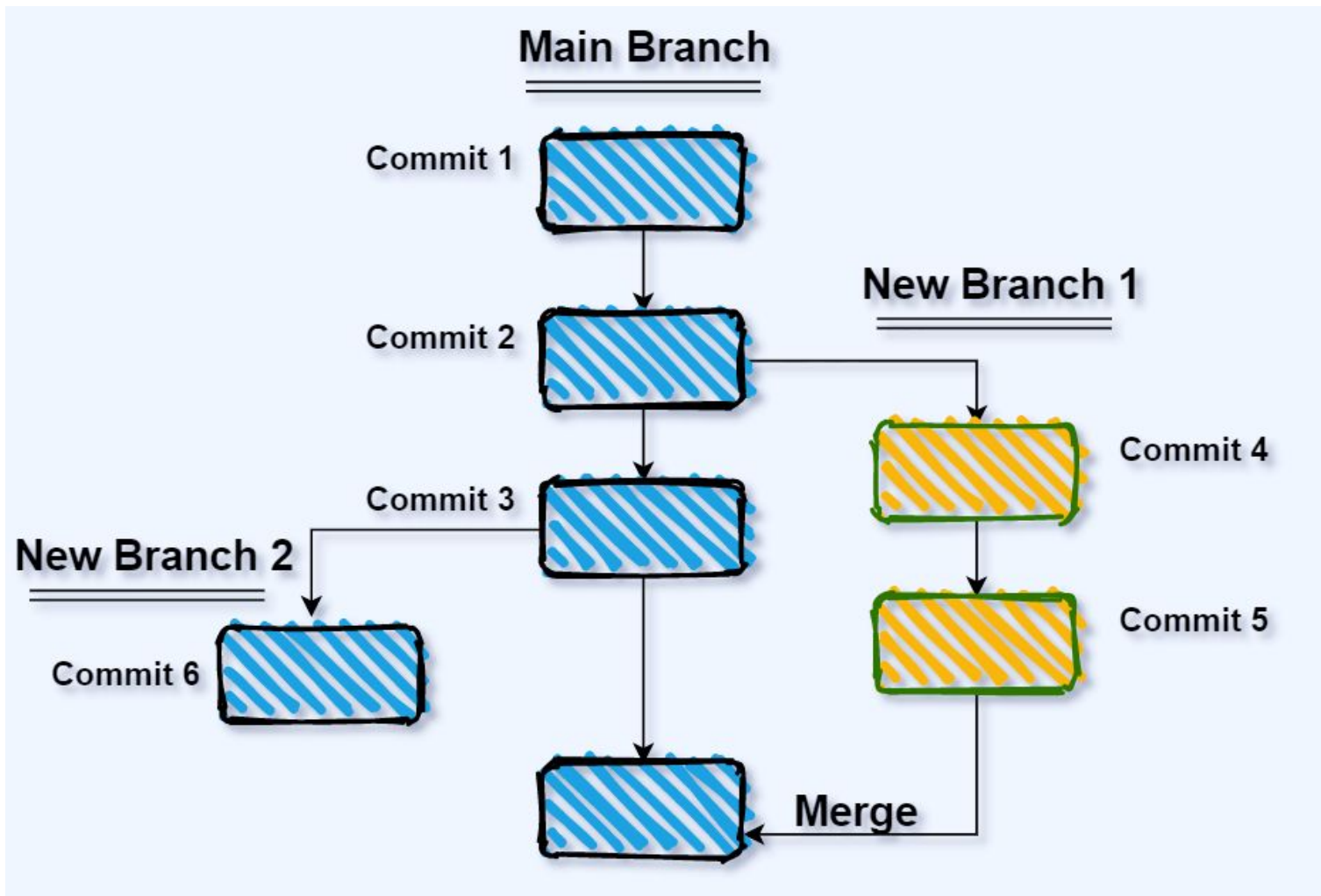




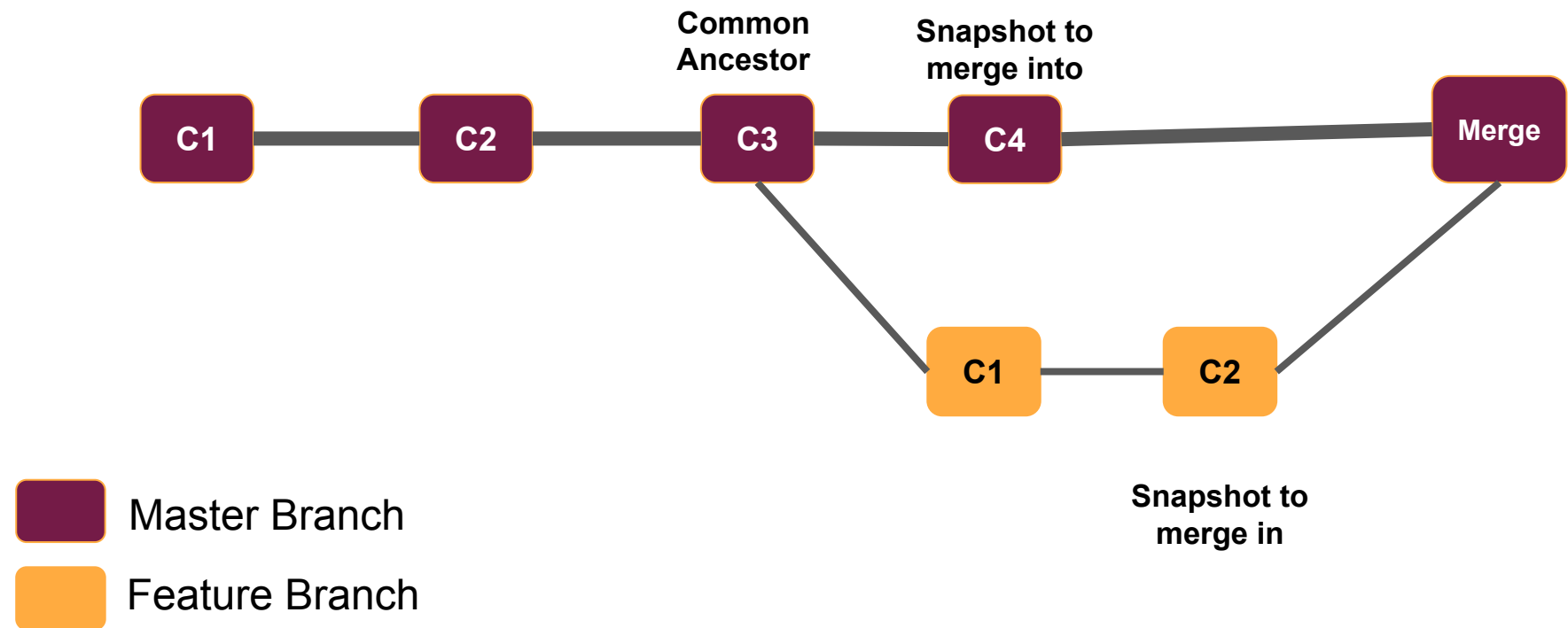
 Master Branch



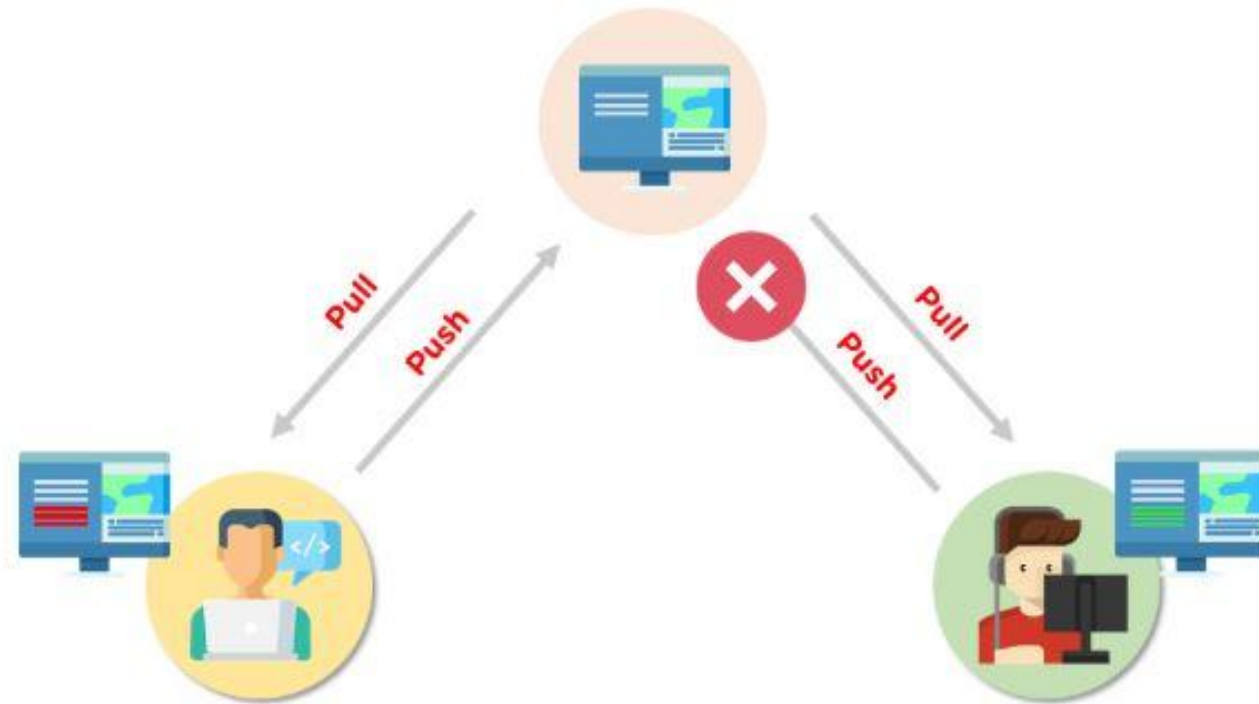




Merging



What is a Git Merge Conflict?



Merge Conflicts

```
graph TD; A[Merge Conflicts] --> B[While starting the merge process]; A --> C[During the merge process]; B --- D[If there are changes in the working directory's stage area for the current project, merging won't start. In this case, conflicts happen due to pending changes that need to be stabilized]; C --- E[When there is a conflict between the local branch and the branch being merged. Git resolves as much as possible, but there are things that have to be resolved manually in the conflicted files];
```

While starting the merge process

If there are changes in the working directory's stage area for the current project, merging won't start.

In this case, conflicts happen due to pending changes that need to be stabilized

During the merge process

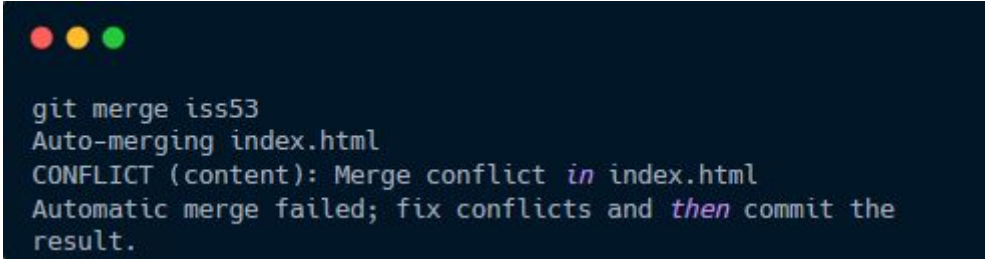
When there is a conflict between the local branch and the branch being merged.

Git resolves as much as possible, but there are things that have to be resolved manually in the conflicted files

Merge Conflicts

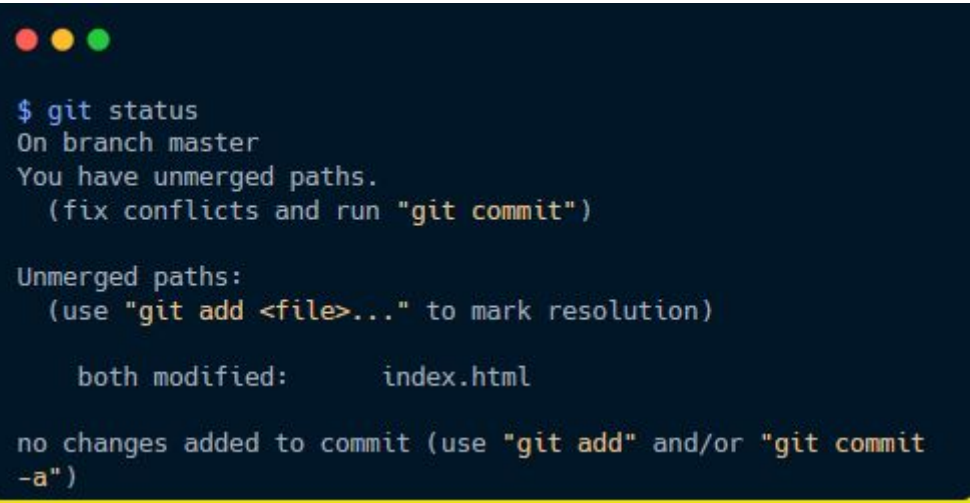
Occasionally, this process doesn't go smoothly.

If you changed the same part of the same file differently in the two branches you're merging, Git won't be able to merge them cleanly. If your fix for feature branch modified the same part of a file as the hotfix branch, you'll get a merge conflict that looks something like this:

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top left corner. The text inside the terminal is white and shows the output of a 'git merge' command.

```
git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the
result.
```

Git hasn't automatically created a new merge commit. It has paused the process while you resolve the conflict. If you want to see which files are unmerged at any point after a merge conflict, you can run `git status`:

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) in the top left corner. The text is displayed in a monospaced font with syntax highlighting: '\$' is blue, 'git' is blue, and 'status', 'master', 'paths.', 'commit', 'paths:', 'resolution)', 'modified:', 'index.html', 'commit', and '-a' are orange. The output shows the current branch is master, there are unmerged paths (index.html), and no changes are added to the commit.

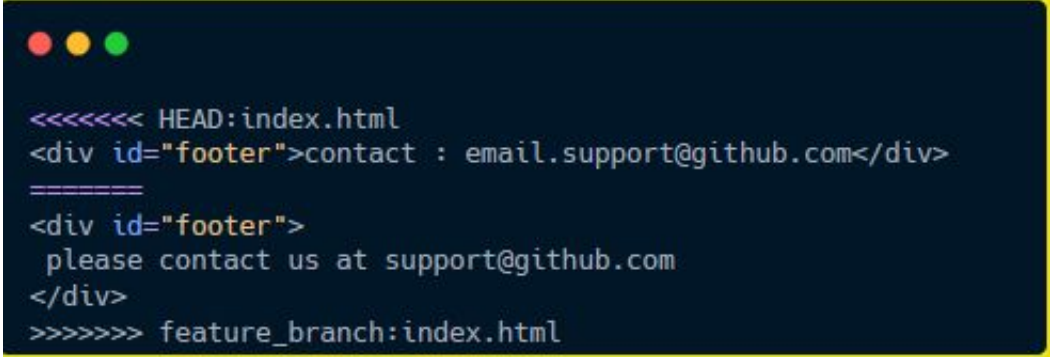
```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:    index.html


no changes added to commit (use "git add" and/or "git commit
-a")
```

Anything that has merge conflicts and hasn't been resolved is listed as unmerged. Git adds standard conflict-resolution markers to the files that have conflicts, so you can open them manually and resolve those conflicts. Your file contains a section that looks something like this:

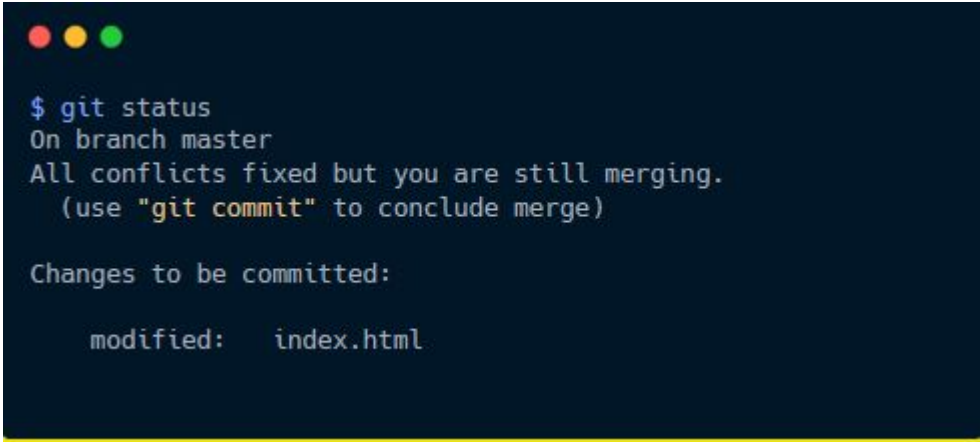


```
<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>> feature_branch:index.html
```

The version in HEAD (your master branch, because that was what you had checked out when you ran your merge command) is the top part of that block (everything above the =====), while the version in your feature_branch looks like everything in the bottom part. In order to resolve the conflict, you have to either choose one side or the other or merge the contents yourself. For instance, you might resolve this conflict by replacing the entire block with this:



```
<div id="footer">
please contact us at email.support@github.com
</div>
```



```
$ git status
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:
  modified:   index.html
```



git commit

Git commands to resolve conflicts

- **git log --merge:** produce the list of commits that are causing the conflict.
- **git diff:** Identify the differences between the states repositories or files.
- **git checkout:** Used to undo the changes made to the file, or for changing branches.
- **git reset --mixed:** Used to undo changes to the working directory and staging area.
- **git merge --abort:** Helps in exiting the merge process and returning back to the state before the merging began.
- **git reset:** Used at the time of merge conflict to reset the conflicted files to their original state.