Introduction to Git

Learn Version Control Fundamentals

Version Control   Collaboration   Branching   History Tracking

What is Git?

Git is a **distributed version control system** that tracks changes in your code over time.

- ✓ Reverts to previous versions
- ✓ Enables team collaboration
- ✓ Provides safety net for changes

Track History

Every commit is a snapshot of your project

Collaborate

Work together without overwriting changes

Branch Safely

Experiment in parallel without breaking main

Getting Started

1 Configure Identity

Set up your name and email for commit attribution

2 Initialize Repository

Create a new Git repository in your project directory

3 Understand .git

Hidden directory storing all version control data

Identity Setup

```
git config --global user.name "Your Name"
git config --global user.email
"email@example.com"
```

Repository Initialization

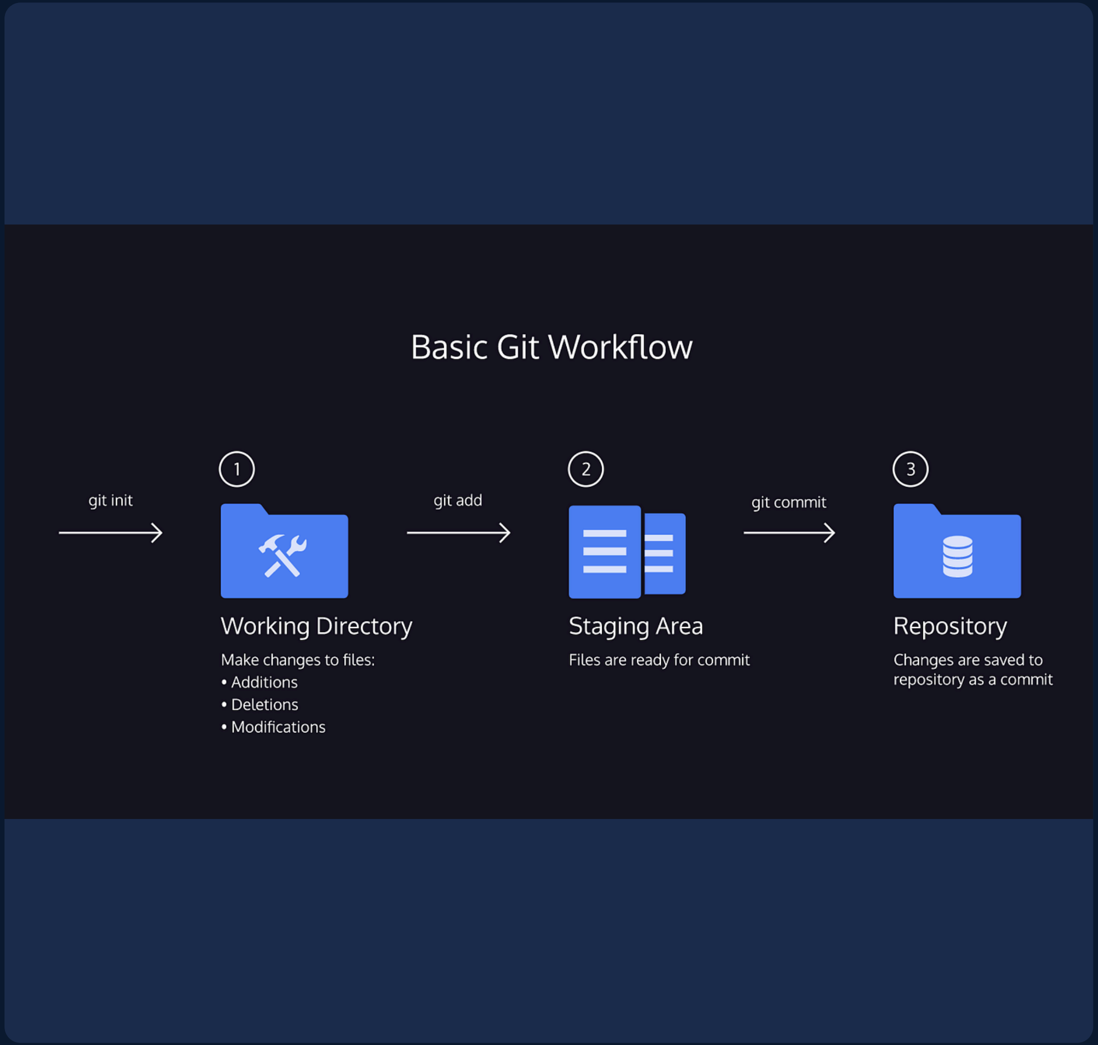
```
git init
```

Creates .git directory in project root

The .git Directory

Contains commit history, branch references, and configuration. Git manages this automatically - never edit directly!

The Git Workflow



Working Directory

Your actual files being edited

Staging Area

Prepared files for next commit

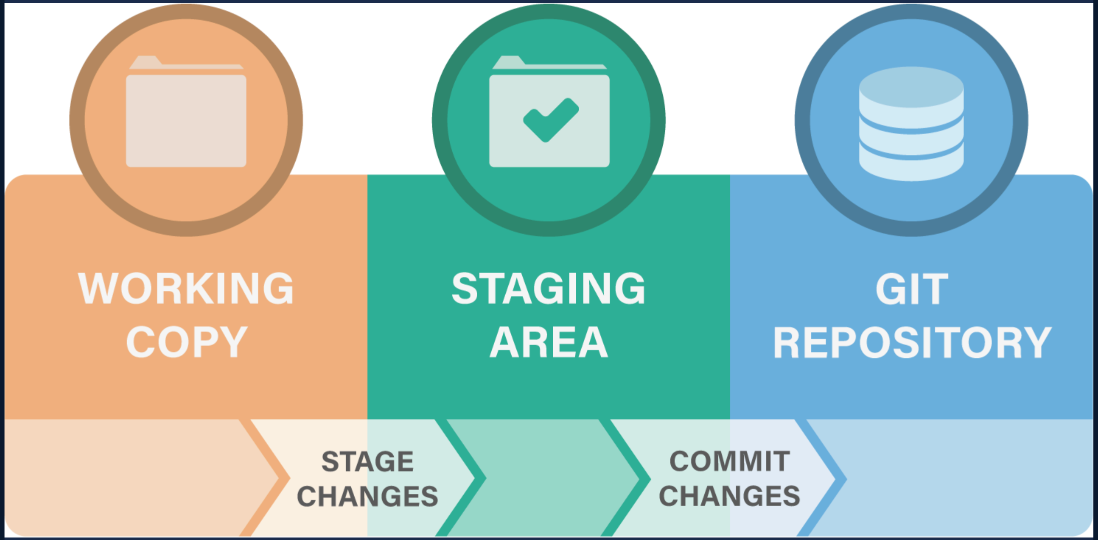
Command: `git add`

Local Repository

Permanent history of changes

Command: `git commit`

Track and Stage Changes



Workflow

Working Directory → Staging Area → Repository

Check Status

```
git status
```

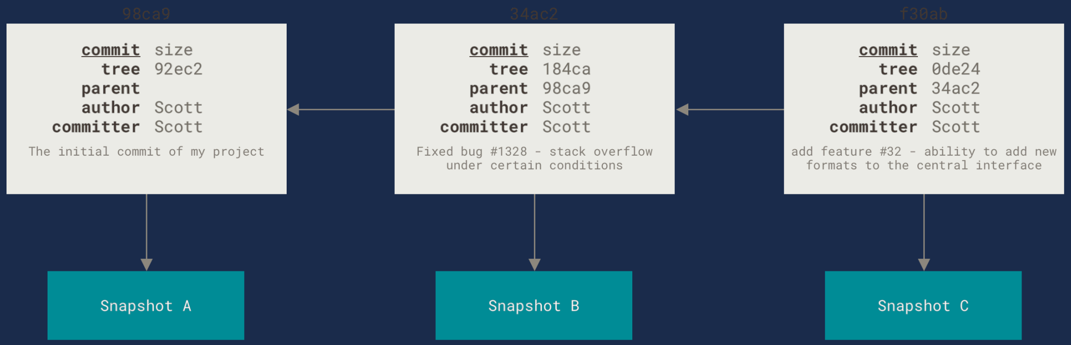
Stage Specific File

```
git add filename
```

Stage All Changes

```
git add .
```

Understanding Commits



Permanent Snapshot

Each commit saves a complete snapshot of your project at that moment in time

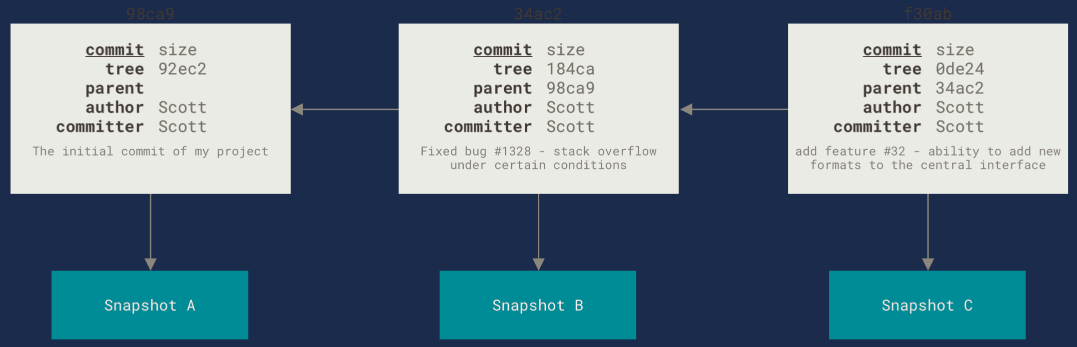
Unique Hash ID

Every commit gets a unique 40-character ID that identifies it forever

History Chain

Commits point to their parent, forming an unbreakable chain of history

Create Commit Snapshots



What is a Commit?

A permanent snapshot of your project at a specific point in time, containing all staged changes with a descriptive message.

Create Commit

```
git commit -m "Descriptive message"
```

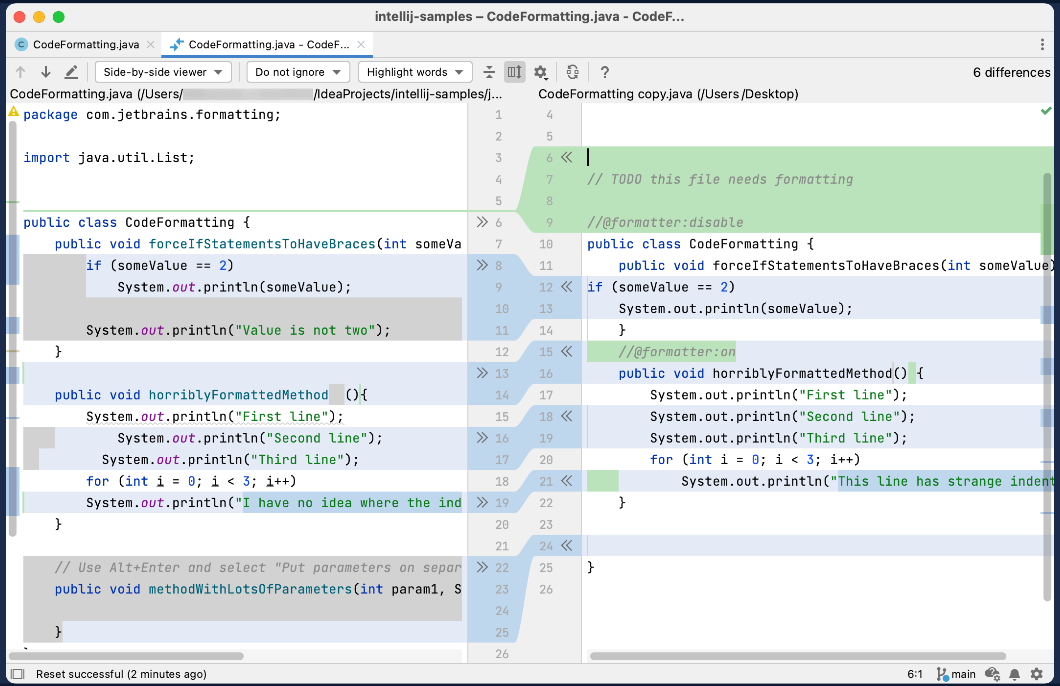
View Commit History

```
git log --oneline
```

View Detailed History

```
git log
```

Compare Changes



CodeFormatting.java (/Users/[redacted]/IdeaProjects/intellij-samples/j...

```
package com.jetbrains.formatting;

import java.util.List;


public class CodeFormatting {
    public void forceIfStatementsToHaveBraces(int someVa
        if (someValue == 2)
            System.out.println(someValue);

        System.out.println("Value is not two");
    }

    public void horriblyFormattedMethod  (){
        System.out.println("First line");
            System.out.println("Second line");
          System.out.println("Third line");
        for (int i = 0; i < 3; i++)
        System.out.println("I have no idea where the ind
    }

    // Use Alt+Enter and select "Put parameters on separ
    public void methodWithLotsOfParameters(int param1, S

    }
```

CodeFormatting copy.java (/Users /Desktop)

```


// TODO this file needs formatting

//@formatter:disable
public class CodeFormatting {
    public void forceIfStatementsToHaveBraces(int someValue)
if (someValue == 2)
    System.out.println(someValue);
    }
    //@formatter:on
    public void horriblyFormattedMethod() {
        System.out.println("First line");
        System.out.println("Second line");
        System.out.println("Third line");
        for (int i = 0; i < 3; i++)
            System.out.println("This line has strange indent
    }


}
```

Reset successful (2 minutes ago)   6:1   main

Why Compare?

Review what changed before committing. See exactly which lines were added, modified, or deleted in your files.

Working vs Staged

`git diff`

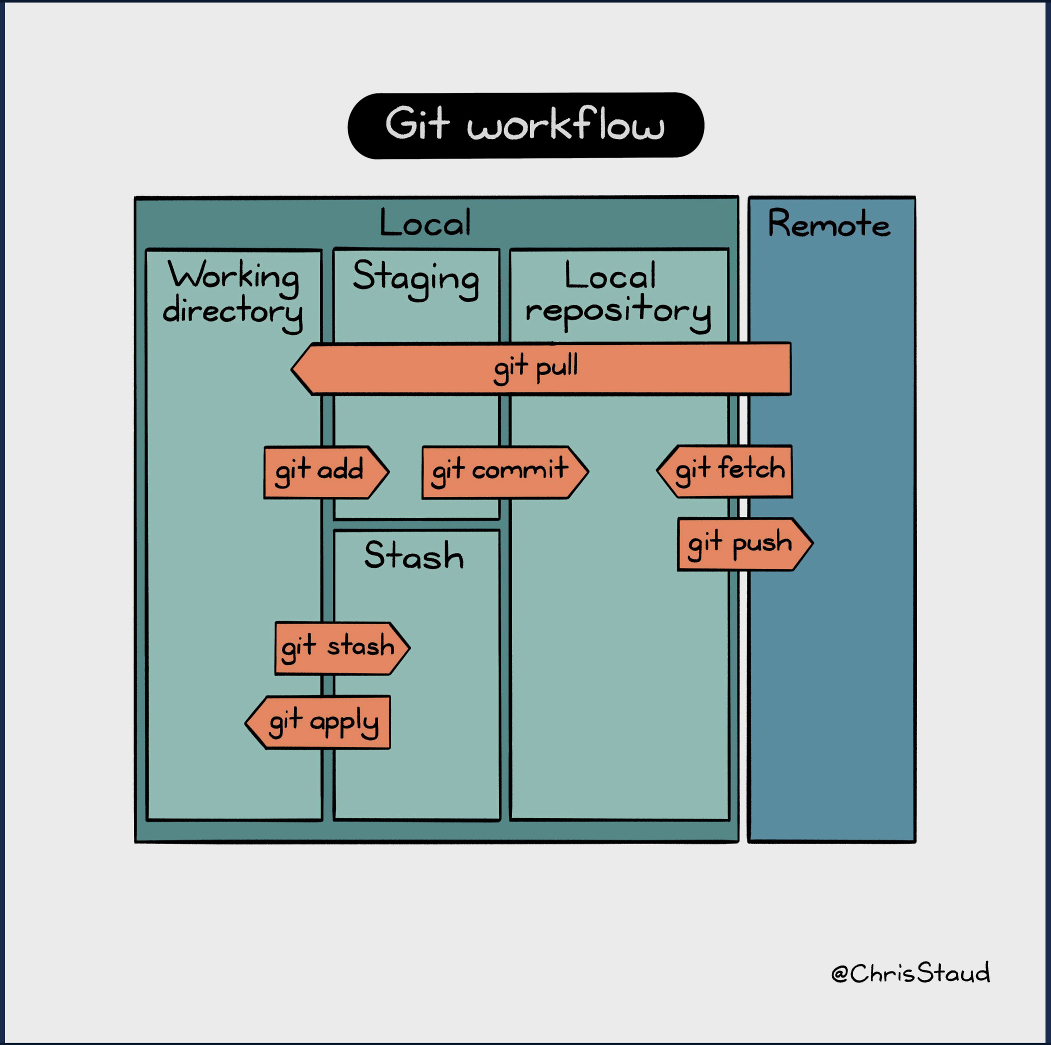
Staged vs Last Commit

`git diff --staged`

Compare with Commit

`git diff commit-hash`

Remote Repositories



Push to Remote

Upload your commits from local to remote repository

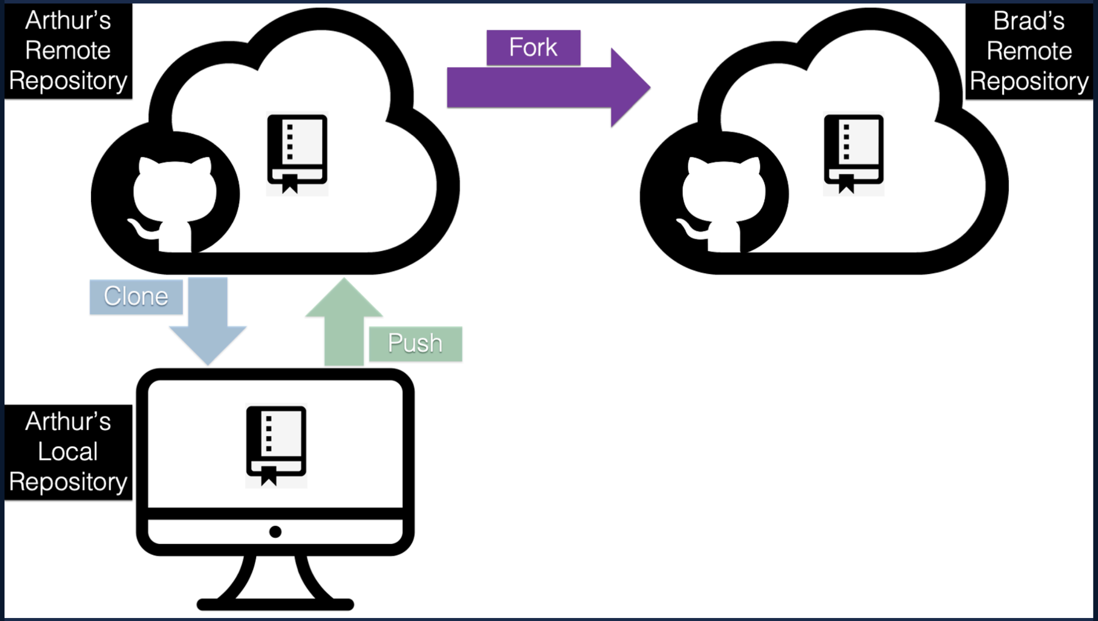
Pull from Remote

Download updates from remote to your local repository

Team Collaboration

Share work, review code, and collaborate with team members

Clone Remote Repository



Why Clone?

Download a complete copy of a remote repository with all its history. Perfect for starting work on existing projects or contributing to open source.

Clone Repository

```
git clone
https://github.com/user/repo.git
```

Clone to Specific Folder

```
git clone url folder-name
```

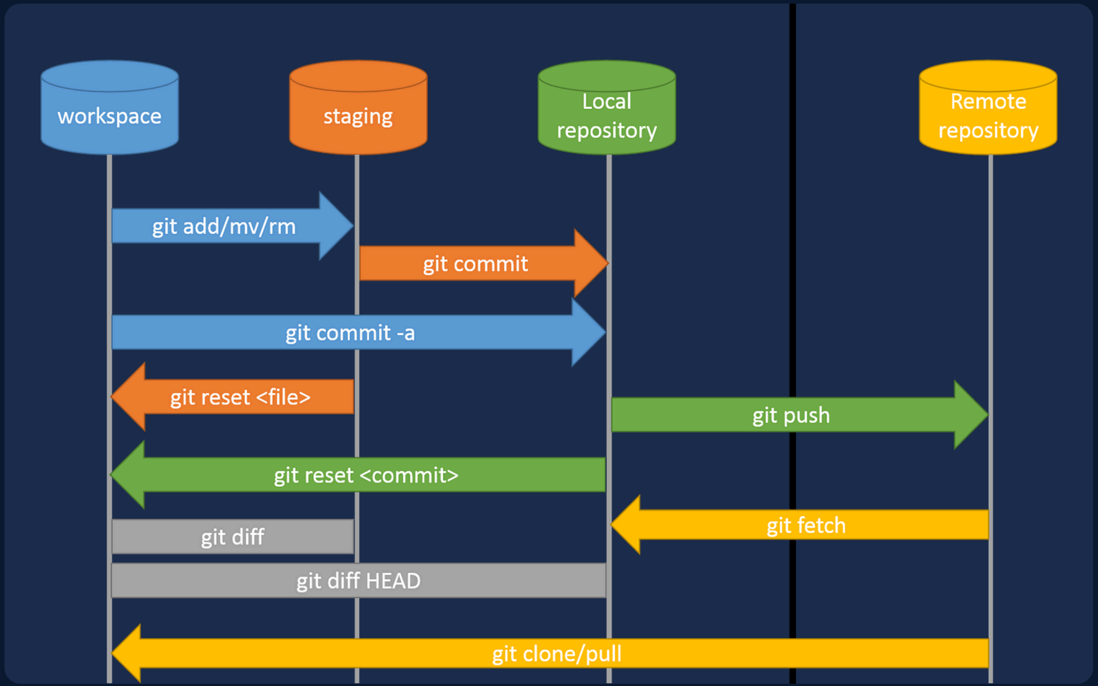
View Remote Repositories

```
git remote -v
```

Add Remote Repository

```
git remote add origin url
```

Sync with Remote



Why Sync?

Upload your work to share with the team and download updates from others. Keeps your local repository in sync with the remote.

Push to Remote

```
git push origin branch-name
```

Pull from Remote

```
git pull origin branch-name
```

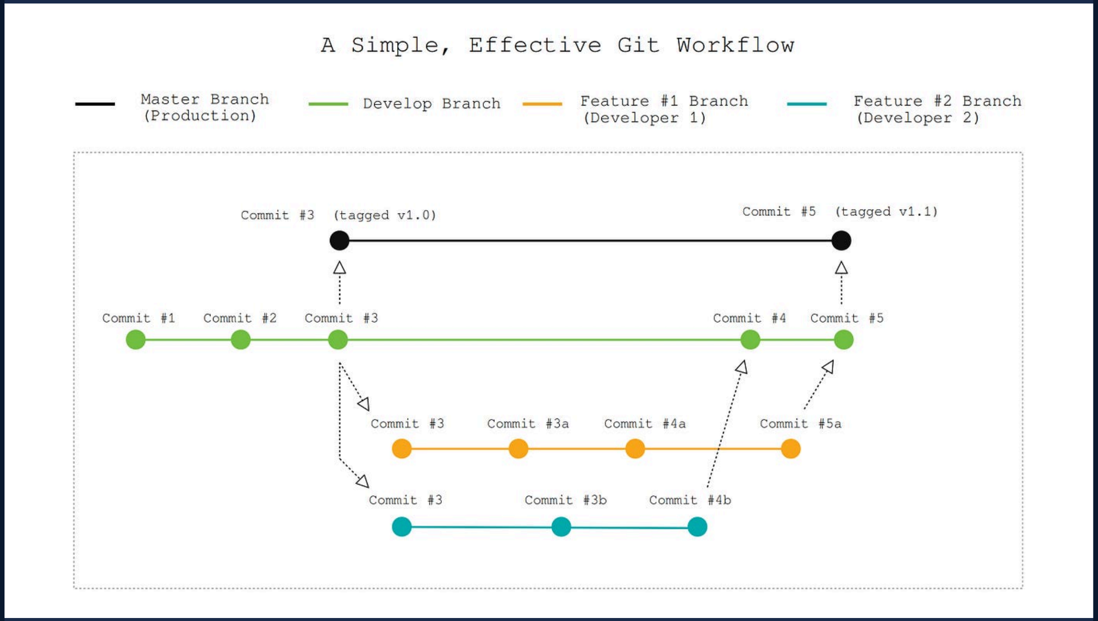
Fetch Changes

```
git fetch origin
```

Pull with Rebase

```
git pull --rebase origin branch-name
```

Branching Concept



Parallel Development

Work on multiple features simultaneously

Safe Experimentation

Test features without affecting main

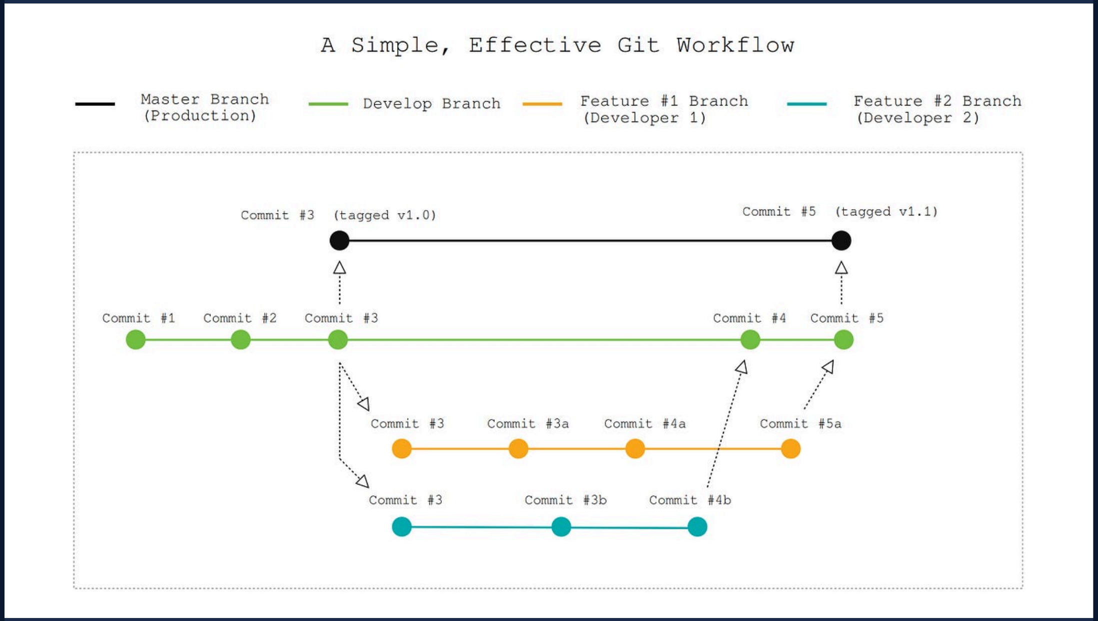
Merge When Ready

Combine work back into main branch

Team Collaboration

Each developer works on their own branch

Create and Switch Branches



List All Branches

```
git branch
```

Create New Branch

```
git branch new-feature
```

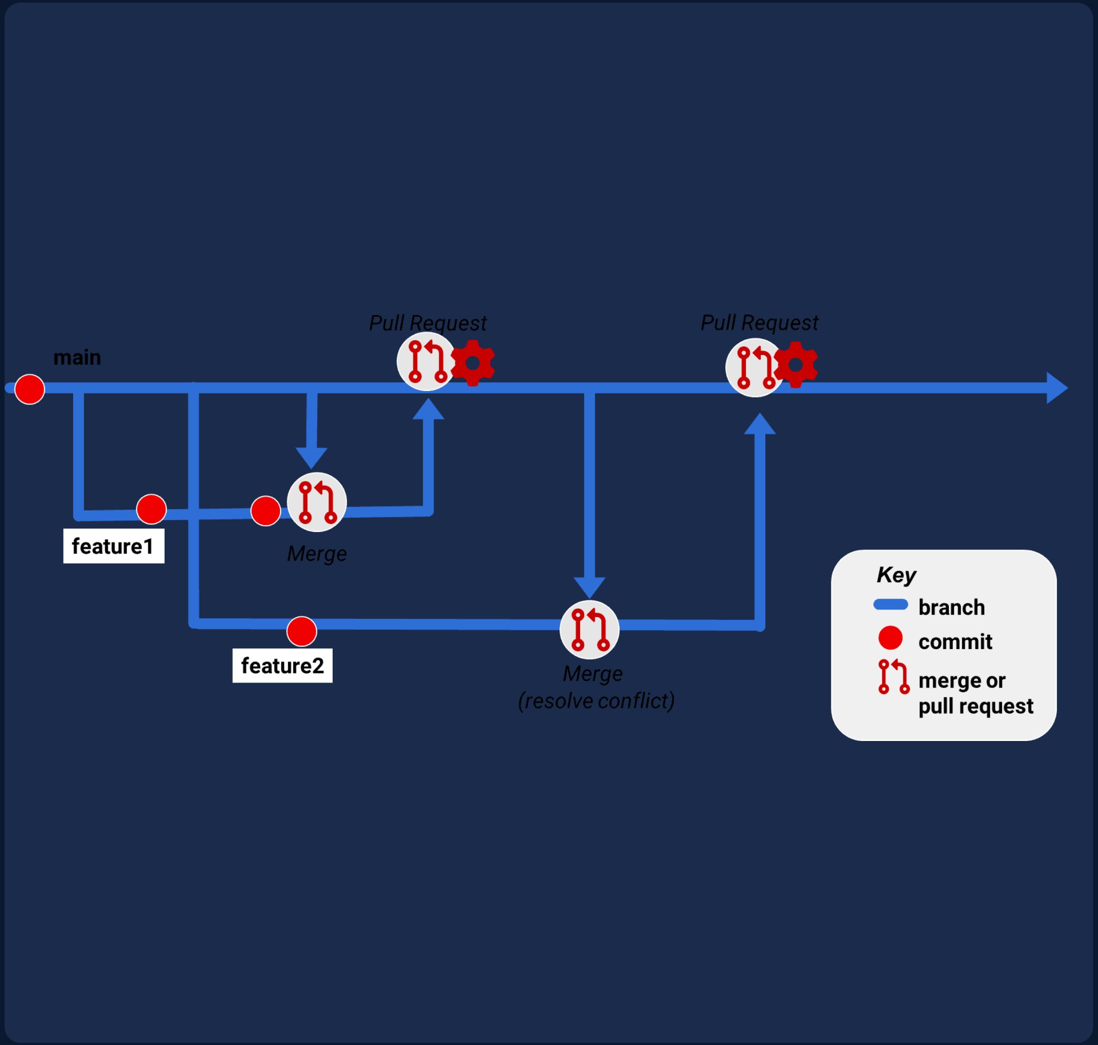
Switch to Branch

```
git checkout new-feature
```

Why Branch?

Create parallel workspaces to develop features independently. Switch between branches without affecting each other.

Pull Request Workflow



Feature Branch

Create and develop your feature

↓

Create PR

Propose changes to main branch

↓

Team Review

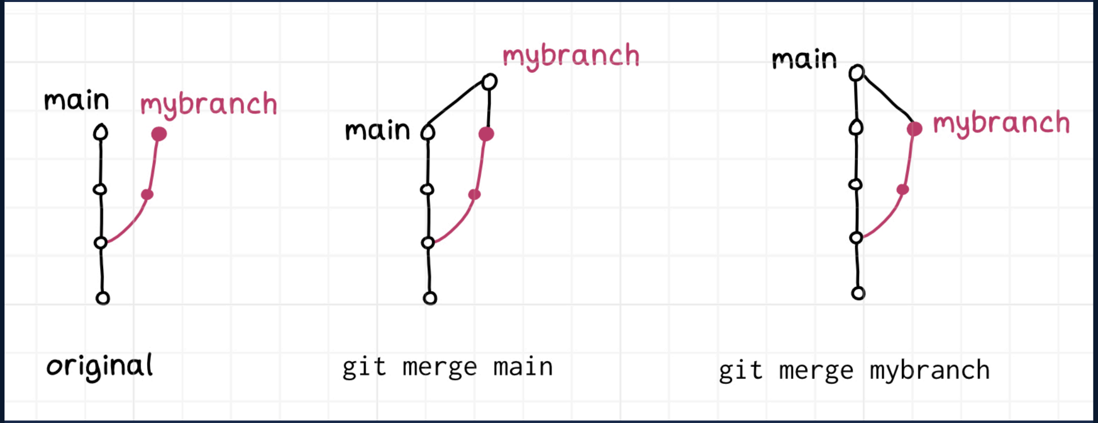
Code review and discussion

↓

Merge to Main

Combine approved changes

Merge Branches



Why Merge?

Combine work from feature branches back into main.
Creates a merge commit that preserves both histories.

Merge into Current Branch

```
git merge branch-name
```

Visualize Merge History

```
git log --graph
```

View All Branches

```
git branch -a
```

Delete Merged Branch

```
git branch -d branch-name
```

Best Practices

Commit Messages

- ✓ Use present tense: "Add feature"
- ✓ Be descriptive & concise
- ✓ First line ≤ 50 characters

Branch Naming

```
feature/description
bugfix/description
hotfix/description
```

Collaboration

- ✓ Review code before merging
- ✓ Keep branches focused
- ✓ Clean up merged branches

Quality Standards

- ✓ Write tests for new features
- ✓ Update documentation