# Introduction to Git

Git is a distributed version control system created in 2005 by Linus Torvalds, the creator of Linux. Originally developed to manage the Linux kernel source code, Git is now maintained by Junio Hamano and has become the industry standard for collaborative software development projects worldwide.

by Naresh Chaurasia

# What is Version Control?

Version control is a system that records changes to files over time so you can recall specific versions later. It serves as the foundation of collaborative development by:

- Tracking modifications to code and documents
- Creating a historical record of all changes
- Enabling multiple developers to work simultaneously
- Providing mechanisms to revert to previous versions

# Types of Version Control Systems

1

## Centralised VCS

Systems like Subversion (SVN) maintain a single central repository on a server. Developers check out files to work on them and commit changes back to the central repository.

- Single source of truth
- Simpler model to understand
- Dependent on server availability

2

## Distributed VCS

Git and similar systems provide every user with a complete copy of the repository, including its full history. This enables offline work and improves reliability.

- Works offline with local operations
- Faster performance for most tasks
- Greater flexibility in workflows

# Why Use Git?

## Performance

Git operations are lightning-fast because they run locally. Most operations need no network access, making Git significantly faster than centralised systems.

## Flexibility

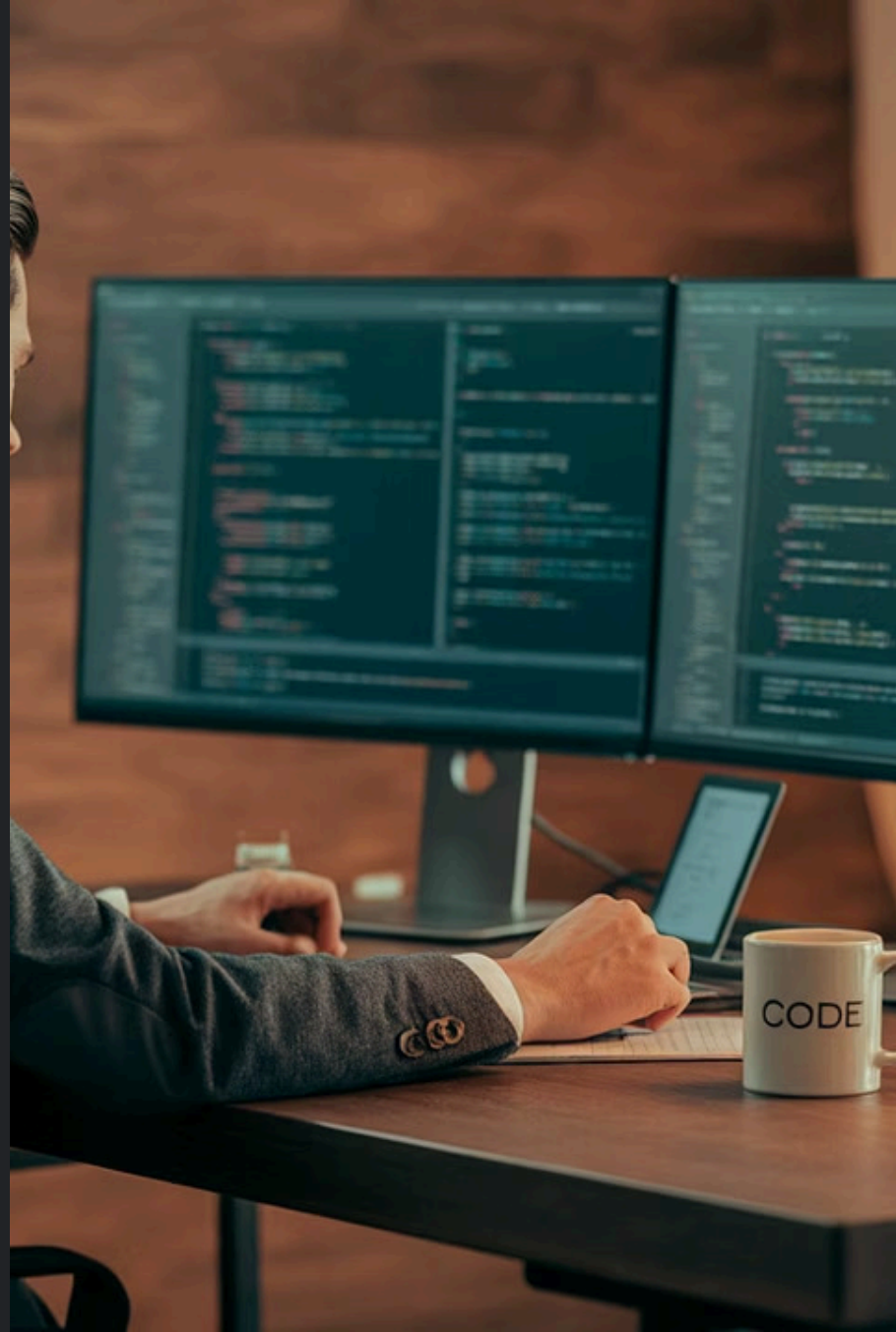
Git's branching model makes it easy to support parallel development, feature isolation, and experimental work without affecting the main codebase.

## Data Integrity

Git uses SHA-1 checksums to verify data integrity. Content is addressed by its hash, making it impossible to change anything without Git knowing.

## Safety

With Git, it's difficult to lose work. Changes are tracked meticulously, and the distributed nature means multiple backups exist naturally.



# Git Workflow Overview

## Working Directory

This is where you make changes to your files. It's your project's actual filesystem on your machine where you edit code.

## Staging Area

Also called the "index," this is a preparation area where you select which changes to include in your next commit.

## Local Repository

Contains all committed snapshots of your project. Stored in the .git directory, this holds your project's complete history.

## Remote Repository

A shared copy of your repository hosted on a server like GitHub or GitLab, enabling collaboration between team members.

# Basic Git Commands

```
# Create a new repository
git init

# Clone an existing repository
git clone
https://github.com/user/repo.git

# Check file status
git status

# Stage changes for commit
git add filename
git add .  # stage all changes
```

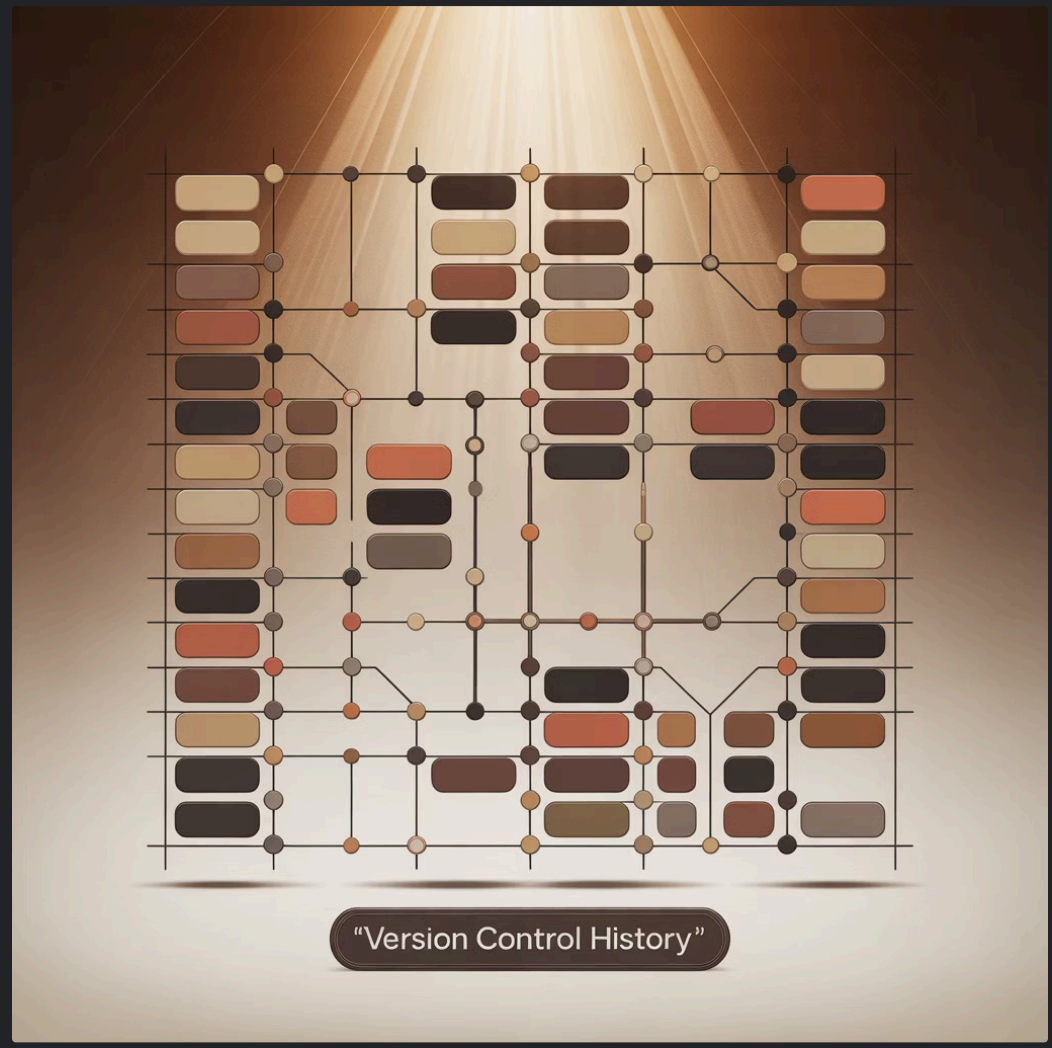
```
# Commit staged changes
git commit -m "Descriptive
message"

# Push changes to remote
git push origin main

# Pull remote changes
git pull origin main

# View commit history
git log
```

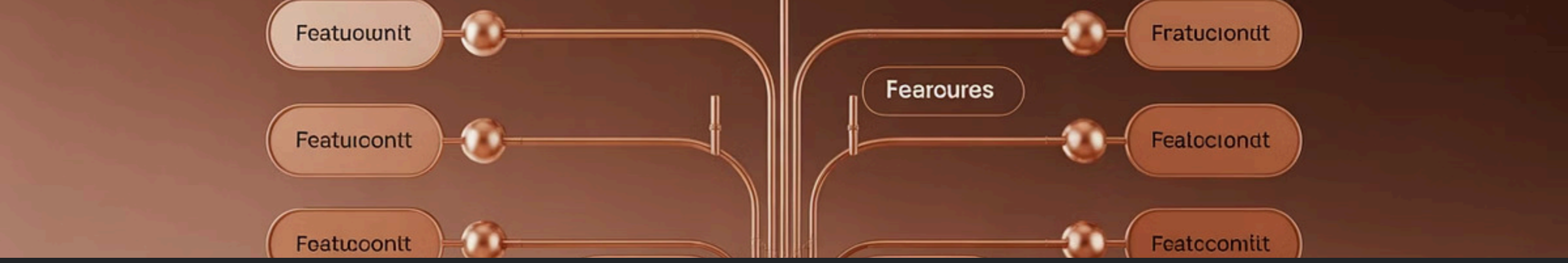
# Understanding Commits

A commit in Git represents a snapshot of your project at a specific point in time. Each commit:

- Has a unique SHA-1 hash identifier (40-character hexadecimal string)
- Contains metadata: author, timestamp, message
- Points to its parent commit(s), forming a directed acyclic graph
- Preserves the exact state of all tracked files

Good commit messages are crucial for tracking history and understanding changes. They should be concise yet descriptive, explaining **why** a change was made rather than just **what** changed.






# Branching in Git

## 1 What is a Branch?

A branch is simply a lightweight movable pointer to a commit. When you create a branch, Git simply creates a new pointer—it doesn't change the repository in any other way.

## 2 Working with Branches

```
# Create a new branch
git branch feature-name

# Switch to a branch
git checkout feature-name
# or (newer syntax)
git switch feature-name

# Create and switch in one command
git checkout -b feature-name
```

## 3 Branch Management

```
# List all branches
git branch

# Delete a branch
git branch -d feature-name

# Push branch to remote
git push origin feature-name
```

# Merging and Conflicts

## Merging Branches

```
# Switch to target branch
git checkout main

# Merge a branch into current branch
git merge feature-branch
```

Git performs two types of merges:

- **Fast-forward:** When the target branch is a direct ancestor
- **Three-way merge:** When branches have diverged

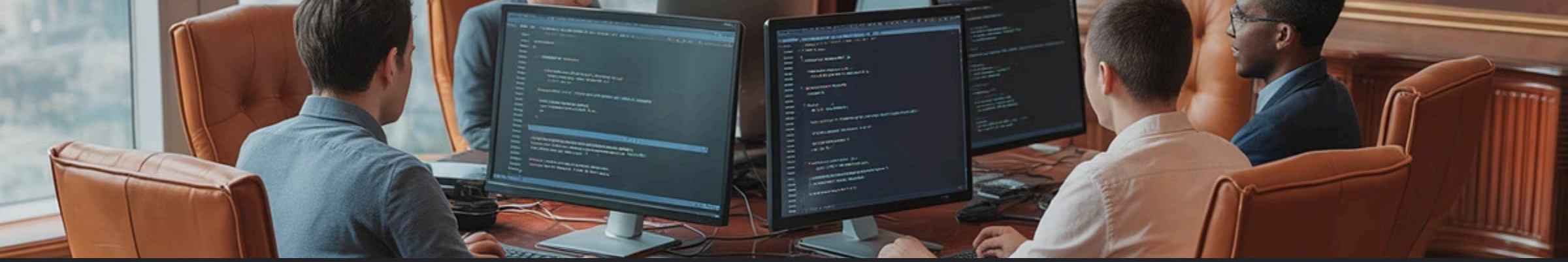
## Resolving Conflicts

Conflicts occur when the same part of a file is changed differently in the branches being merged. Git marks these in the file:

```
<<<<<<< HEAD
Current branch code
=======
Incoming branch code
>>>>>>> feature-branch
```

To resolve:

1. Edit files to fix conflicts
2. Stage resolved files with `git add`
3. Complete the merge with git commit



# Remote Repositories and Collaboration

## Working with Remotes

```
# List remote repositories
git remote -v

# Add a remote
git remote add origin
https://github.com/user/repo.git

# Remove a remote
git remote remove origin
```

## Syncing with Remotes

```
# Fetch remote changes without
merging
git fetch origin

# Pull changes (fetch + merge)
git pull origin main

# Push local changes to remote
git push origin main
```

## Collaboration Models

- **Centralised Workflow:** Everyone works on main branch
- **Feature Branch Workflow:** New feature = new branch
- **Gitflow:** Structured branching model
- **Forking Workflow:** Fork repository, work independently

# Practical Example Workflow

## 1. Clone the Repository

```
git clone https://github.com/user/project.git
cd project
```

## 2. Create a Feature Branch

```
git checkout -b feature-login-page
```

## 3. Make Changes and Commit

```
# Make changes to files
git status
git add .
git commit -m "Add login form and authentication"
```

## 4. Push Branch to Remote

```
git push -u origin feature-login-page
```

## 5. Create Pull Request

Use GitHub/GitLab interface to create PR

## 6. Merge After Review

```
git checkout main
git pull origin main
git merge feature-login-page
git push origin main
```

# Summary and Next Steps

## Key Concepts Covered

- Git fundamentals and advantages
- Working with repositories and commits
- Branching and merging workflows
- Collaboration using remote repositories
- Resolving conflicts and managing changes

## Recommended Resources

- Git official documentation: git-scm.com/doc
- GitHub Learning Lab: lab.github.com
- "Pro Git" book by Scott Chacon (free online)