Git Branching Complete Lab Guide

Table of Contents

Introduction to Git Branching {#introduction}

What is Branching?

Branching in Git allows you to create separate lines of development in your project. Think of it as creating a copy of your project where you can experiment without affecting the main codebase.

Why Use Branches?

- **Feature Development**: Work on new features without breaking main code
- **Bug Fixes**: Fix issues in isolation
- **Experimentation**: Try new ideas safely
- **Collaboration**: Multiple developers can work simultaneously

Key Concepts

- **Branch**: A movable pointer to a specific commit
- **HEAD**: Points to the current branch you're working on
- **Master/Main**: The default primary branch
- **Merge**: Combining changes from different branches

Lab Setup {#lab-setup}

Prerequisites

- Git installed on your system
- Basic understanding of Git commands
- Text editor (VS Code, Notepad++, etc.)

Initial Setup

Step 1: Create Project Directory

```bash
mkdir git-branching-lab
cd git-branching-lab
```

Step 2: Initialize Git Repository

```bash
git init
```

Step 3: Configure Git (if not done globally)

```bash
git config user.name "Your Name"
git config user.email "your.email@example.com"
```

Step 4: Create Initial HTML File Create a file named `index.html`:

```html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Git Branching Lab</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            max-width: 800px;
            margin: 0 auto;
            padding: 20px;
        }
        .header {
            background-color: #f0f0f0;
            padding: 20px;
            text-align: center;
        }
    </style>
</head>
<body>
    <div class="header">
        <h1>Welcome to Git Branching Lab</h1>
        <p>This is our main branch content</p>
    </div>

    <main>
        <h2>Project Features</h2>
        <ul>
            <li>Basic HTML Structure</li>
        </ul>
    </main>
</body>
</html>
```

Step 5: Make Initial Commit

```
bash

git add index.html
git commit -m "Initial commit: Add basic HTML structure"
```

Creating Branches {#creating-branches}

Method 1: Create Branch Only

```bash
# Create a new branch (doesn't switch to it)
git branch branch-name
```

Method 2: Create and Switch

```bash
# Create and switch to new branch
git checkout -b branch-name
```

Method 3: Modern Approach (Git 2.23+)

```bash
# Create and switch using git switch
git switch -c branch-name
```

Lab Exercise: Creating Your First Branch

```bash
# Check current branch
git branch

# Create a feature branch
git checkout -b feature/navigation

# Verify branch creation
git branch
```

Switching Between Branches {#switching-branches}

Using git checkout

```bash
# Switch to existing branch
git checkout branch-name

# Switch back to main
git checkout main
```

Using git switch (Recommended)

```bash
# Switch to existing branch
git switch branch-name

# Switch back to main
git switch main
```

Lab Exercise: Branch Switching

```bash
# Create multiple branches
git checkout -b feature/styling
git checkout -b feature/content
git checkout -b bugfix/header

# Practice switching
git switch main
git switch feature/navigation
git switch feature/styling

# Check current branch
git branch --show-current
```

Working with Branches {#working-with-branches}

Viewing Branches

```bash
# List local branches
git branch

# List all branches (local and remote)
git branch -a

# List remote branches
git branch -r

# Show branch with last commit
git branch -v
```

Branch Status and Information

```bash
```

```
# Show current branch
git status

# Show branch history
git log --oneline --graph --all

# Show branch relationships
git show-branch
```

Lab Exercises {#lab-exercises}

Exercise 1: Navigation Feature Branch

Step 1: Create and Switch to Feature Branch

```bash
git switch main
git checkout -b feature/navigation
```

Step 2: Add Navigation to HTML Modify `index.html` to add navigation:

```html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Git Branching Lab</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            max-width: 800px;
            margin: 0 auto;
            padding: 20px;
        }
        .header {
            background-color: #f0f0f0;
            padding: 20px;
            text-align: center;
        }
        nav {
            background-color: #333;
            padding: 10px;
        }
        nav ul {
            list-style: none;
            margin: 0;
            padding: 0;
            display: flex;
        }
        nav li {
            margin-right: 20px;
        }
        nav a {
            color: white;
            text-decoration: none;
        }
    </style>
</head>
<body>
    <nav>
        <ul>
            <li><a href="#home">Home</a></li>
            <li><a href="#about">About</a></li>
            <li><a href="#contact">Contact</a></li>
        </ul>
    </nav>
```

```
    <div class="header">
        <h1>Welcome to Git Branching Lab</h1>
        <p>This is our main branch content</p>
    </div>

    <main>
        <h2>Project Features</h2>
        <ul>
            <li>Basic HTML Structure</li>
            <li>Navigation Menu</li>
        </ul>
    </main>
</body>
</html>
```

Step 3: Commit Changes

```bash
git add index.html
git commit -m "Add navigation menu to header"
```

Exercise 2: Content Feature Branch

Step 1: Switch to Main and Create New Branch

```bash
git switch main
git checkout -b feature/content
```

Step 2: Add Content Section Modify `index.html` to add more content:

```html
<!-- Add this section before closing </body> tag -->
<section id="about">
    <h2>About This Project</h2>
    <p>This project demonstrates Git branching concepts through practical exercises.</p>
</section>

<section id="contact">
    <h2>Contact Information</h2>
    <p>Email: student@example.com</p>
</section>
```

Step 3: Update Features List

```html
<main>
    <h2>Project Features</h2>
    <ul>
        <li>Basic HTML Structure</li>
        <li>Content Sections</li>
        <li>About Page</li>
        <li>Contact Information</li>
    </ul>
</main>
```

Step 4: Commit Changes

```bash
git add index.html
git commit -m "Add about and contact sections"
```

Exercise 3: Styling Feature Branch

Step 1: Create Styling Branch

```bash
git switch main
git checkout -b feature/styling
```

Step 2: Enhance CSS Styles Add these styles to the `<style>` section:

```css
section {
    margin: 30px 0;
    padding: 20px;
    border-left: 4px solid #007bff;
    background-color: #f8f9fa;
}

.features {
    background-color: #e7f3ff;
    padding: 15px;
    border-radius: 5px;
}
```

Step 3: Update HTML Structure

```html
<main class="features">
  <h2>Project Features</h2>
  <ul>
    <li>Basic HTML Structure</li>
    <li>Enhanced Styling</li>
    <li>Responsive Design</li>
  </ul>
</main>
```

Step 4: Commit Changes

```bash
git add index.html
git commit -m "Add enhanced styling and layout"
```

Merging Branches {#merging-branches}

Types of Merges

1. Fast-Forward Merge

When target branch hasn't changed since branch creation.

2. Three-Way Merge

When both branches have new commits.

Merge Commands

```bash
# Switch to target branch (usually main)
git switch main

# Merge feature branch
git merge feature-branch-name

# Merge with commit message
git merge feature-branch-name -m "Merge feature: description"
```

Lab Exercise: Merging Branches

Step 1: Merge Navigation Feature

```bash
git switch main
git merge feature/navigation
```

Step 2: View Results

```bash
git log --oneline --graph
```

Step 3: Merge Content Feature

```bash
git merge feature/content
```

Step 4: Handle Merge Conflicts (if any) If conflicts occur:

```bash
# Edit conflicted files manually
# Remove conflict markers: <<<<<<<, =======, >>>>>>>
# Choose which changes to keep

# After resolving conflicts
git add index.html
git commit -m "Resolve merge conflicts"
```

Step 5: Merge Styling Feature

```bash
git merge feature/styling
```

Deleting Branches {#deleting-branches}

Delete Merged Branches

```bash
```

```
# Delete local branch (safe - only if merged)
git branch -d branch-name

# Force delete branch (dangerous)
git branch -D branch-name
```

Delete Remote Branches

```
bash

# Delete remote branch
git push origin --delete branch-name
```

Lab Exercise: Cleanup Branches

```
bash

# List all branches
git branch

# Delete merged feature branches
git branch -d feature/navigation
git branch -d feature/content
git branch -d feature/styling

# Verify deletion
git branch
```

Advanced Branching Operations {#advanced-operations}

Rebasing (Alternative to Merging)

```
bash

# Switch to feature branch
git switch feature-branch

# Rebase onto main
git rebase main

# Switch to main and merge
git switch main
git merge feature-branch
```

Cherry-picking

```bash
# Apply specific commit from another branch
git cherry-pick commit-hash
```

Branch Renaming

```bash
# Rename current branch
git branch -m new-name

# Rename other branch
git branch -m old-name new-name
```

Lab Exercise: Advanced Operations

Step 1: Create Test Branch

```bash
git checkout -b test/advanced-features
```

Step 2: Make Some Changes

```html
<!-- Add to index.html -->
<footer>
    <p>&copy; 2024 Git Branching Lab. All rights reserved.</p>
</footer>
```

Step 3: Commit and Rebase

```bash
git add index.html
git commit -m "Add footer section"

# Rebase instead of merge
git switch main
git switch test/advanced-features
git rebase main
```

Common Git Commands Reference {#reference}

Branch Management

Command	Description
`git branch`	List local branches
`git branch -a`	List all branches
`git branch branch-name`	Create new branch
`git checkout -b branch-name`	Create and switch to branch
`git switch branch-name`	Switch to branch
`git branch -d branch-name`	Delete merged branch
`git branch -D branch-name`	Force delete branch
`git branch -m new-name`	Rename current branch

Merging and Integration

Command	Description
`git merge branch-name`	Merge branch into current
`git rebase branch-name`	Rebase current branch
`git cherry-pick commit-hash`	Apply specific commit
`git merge --abort`	Abort merge process
`git rebase --abort`	Abort rebase process

Information and Status

Command	Description
`git status`	Show current status
`git branch --show-current`	Show current branch name
`git log --oneline --graph`	Show branch history
`git show-branch`	Show branch relationships
`git diff branch1..branch2`	Compare branches

Lab Assessment Checklist

Basic Operations (Must Complete)

- [ ] Initialize Git repository
- [ ] Create initial HTML file and commit
- [ ] Create at least 3 feature branches
- [ ] Switch between branches successfully
- [ ] Make commits on different branches

- [ ] Merge branches back to main
- [ ] Delete merged branches

Intermediate Operations (Recommended)

- [ ] Handle merge conflicts
- [ ] Use git log to view branch history
- [ ] Rename a branch
- [ ] Create branches from specific commits
- [ ] Use git stash while switching branches

Advanced Operations (Optional)

- [ ] Perform a rebase operation
- [ ] Cherry-pick commits between branches
- [ ] Create and merge pull requests (if using GitHub)
- [ ] Use interactive rebase to squash commits

Troubleshooting Common Issues

Problem: Cannot switch branches with uncommitted changes

Solution:

```bash
# Stash changes
git stash

# Switch branch
git switch branch-name

# Apply stashed changes later
git stash pop
```

Problem: Merge conflicts

Solution:

1. Open conflicted file in text editor
2. Look for conflict markers (`<<<<<<<`, `=======`, `>>>>>>>`)
3. Choose which changes to keep
4. Remove conflict markers
5. Add and commit resolved files

Problem: Accidentally deleted important branch

Solution:

```bash
# Find commit hash of deleted branch
git reflog

# Recreate branch from commit
git checkout -b recovered-branch commit-hash
```

Best Practices

1. **Use Descriptive Branch Names**: `feature/user-authentication` instead of `branch1`
2. **Keep Branches Small**: Focus on single features or fixes
3. **Regular Commits**: Make frequent, logical commits
4. **Clean History**: Use rebase for cleaner project history
5. **Delete Merged Branches**: Keep repository clean
6. **Test Before Merging**: Ensure code works before merging
7. **Use Pull Requests**: For team collaboration and code review

---

Summary

This lab covered the complete Git branching workflow:

- **Creating** branches for feature development
- **Switching** between branches efficiently
- **Working** with multiple branches simultaneously
- **Merging** branches to integrate changes
- **Deleting** branches to maintain clean repository
- **Advanced** operations like rebasing and cherry-picking

Git branching is a powerful feature that enables safe experimentation, parallel development, and organized project management. Practice these concepts regularly to become proficient in Git workflow management.

Lab Completion Time: 2-3 hours **Difficulty Level:** Beginner to Intermediate **Prerequisites:** Basic Git knowledge (init, add, commit, status)