

# Git Branching Strategy & Workflow

---

## Overview

---

This document outlines the comprehensive Git branching strategy, workflow processes, and version control guidelines for the Berean Bible Reading Plan Application.

## Branching Model

---

### Main Branches

#### main

- **Purpose:** Production-ready code
- **Protection:** Protected branch with required reviews
- **Deployment:** Automatically deploys to production
- **Merge Policy:** Only from `develop` via pull request
- **Naming Convention:** `main` (default branch)

#### develop


- **Purpose:** Integration branch for features
- **Protection:** Protected branch with required reviews
- **Deployment:** Automatically deploys to staging environment
- **Merge Policy:** From feature branches and phase branches
- **Naming Convention:** `develop`

### Phase Branches


#### phase-0

- **Purpose:** Project initialization and planning
- **Base:** `main`
- **Scope:** Repository setup, documentation, initial planning
- **Lifecycle:** Long-lived, maintained throughout project

#### phase-1


- **Purpose:** Core MVP development
- **Base:** `develop`
- **Scope:** Basic reading functionality, UI foundation
- **Status:**  Complete (v1.0.0-phase1)

#### phase-2

- **Purpose:** Enhanced features and user experience
- **Base:** `develop`
- **Scope:** Advanced UI, user preferences, enhanced navigation
- **Status:**  Planned

#### phase-3

- **Purpose:** Advanced features and integrations

- **Base:** `develop`
- **Scope:** Social features, analytics, mobile optimization
- **Status:**  Future

## Feature Branches

### Naming Convention

```
feature/<issue-number>-<short-description>
feature/123-add-bookmark-functionality
feature/456-implement-search
```

### Lifecycle

- **Base:** `develop` or relevant phase branch
- **Merge Target:** `develop` or originating phase branch
- **Lifetime:** Short-lived (1-2 weeks maximum)
- **Cleanup:** Deleted after successful merge

## Hotfix Branches

### Naming Convention

```
hotfix/<version>-<short-description>
hotfix/1.0.1-fix-api-timeout
hotfix/1.0.2-security-patch
```

### Lifecycle

- **Base:** `main`
- **Merge Target:** Both `main` and `develop`
- **Lifetime:** Very short-lived (hours to days)
- **Priority:** High priority for production issues

## Release Branches

### Naming Convention

```
release/<version>
release/1.1.0
release/2.0.0-beta
```

### Lifecycle

- **Base:** `develop`
- **Merge Target:** `main` and `develop`
- **Purpose:** Final testing and bug fixes before release
- **Lifetime:** Short-lived (1 week maximum)

## Workflow Process

---

### Feature Development Workflow

#### 1. Create Feature Branch

```
bash
```

```
git checkout develop
git pull origin develop
git checkout -b feature/123-add-bookmark-functionality
```

## 2. Development

```
```bash
# Make changes
git add .
git commit -m "feat(bookmarks): add bookmark save functionality"
```

# Push regularly

```
git push origin feature/123-add-bookmark-functionality
```
```

## 1. Keep Updated

```
bash
# Regularly sync with develop
git checkout develop
git pull origin develop
git checkout feature/123-add-bookmark-functionality
git rebase develop
```

## 2. Create Pull Request

- Target: develop branch
- Include: Description, testing notes, screenshots
- Assign: Reviewers and labels
- Link: Related issues

## 3. Code Review Process

- Automated checks must pass
- At least 1 reviewer approval required
- Address feedback and update branch
- Squash commits if necessary

## 4. Merge and Cleanup

```
bash
# After PR approval and merge
git checkout develop
git pull origin develop
git branch -d feature/123-add-bookmark-functionality
git push origin --delete feature/123-add-bookmark-functionality
```

# Release Workflow

## 1. Create Release Branch

```
bash
git checkout develop
git pull origin develop
git checkout -b release/1.1.0
```

## 2. Prepare Release

```
```bash
```

```

# Update version numbers
npm version 1.1.0 --no-git-tag-version

# Update CHANGELOG.md
# Final testing and bug fixes only
git commit -m "chore(release): prepare v1.1.0"
```

```

### 1. Merge to Main

```

bash
git checkout main
git pull origin main
git merge --no-ff release/1.1.0
git tag -a v1.1.0 -m "Release version 1.1.0"
git push origin main --tags

```

### 2. Merge Back to Develop

```

bash
git checkout develop
git merge --no-ff release/1.1.0
git push origin develop

```

### 3. Cleanup

```

bash
git branch -d release/1.1.0
git push origin --delete release/1.1.0

```

## Hotfix Workflow

### 1. Create Hotfix Branch

```

bash
git checkout main
git pull origin main
git checkout -b hotfix/1.0.1-fix-api-timeout

```

### 2. Fix and Test

```

```bash
# Make minimal fix
git commit -m "fix(api): increase timeout for ESV API calls"

```

```

# Update version
npm version patch --no-git-tag-version
git commit -m "chore(release): bump version to 1.0.1"
```

```

### 1. Merge to Main

```

bash
git checkout main
git merge --no-ff hotfix/1.0.1-fix-api-timeout
git tag -a v1.0.1 -m "Hotfix version 1.0.1"
git push origin main --tags

```

### 2. Merge to Develop

```

bash

```

```
git checkout develop
git merge --no-ff hotfix/1.0.1-fix-api-timeout
git push origin develop
```

## Commit Conventions

### Conventional Commits Format

```
<type>[optional scope]: <description>

[optional body]

[optional footer(s)]
```

### Commit Types

- **feat**: New feature
- **fix**: Bug fix
- **docs**: Documentation changes
- **style**: Code style changes (formatting, etc.)
- **refactor**: Code refactoring
- **test**: Adding or modifying tests
- **chore**: Maintenance tasks
- **perf**: Performance improvements
- **ci**: CI/CD changes
- **build**: Build system changes

### Examples

```
feat(reading-plan): add progress tracking functionality
fix(api): handle ESV API rate limiting errors
docs(readme): update installation instructions
style(components): format code with prettier
refactor(utils): extract common Bible reference parsing
test(components): add unit tests for BibleVerse component
chore(deps): update dependencies to latest versions
perf(api): implement caching for Bible passages
ci(github): add automated testing workflow
build(docker): optimize production Docker image
```

### Scope Guidelines

- **api**: API-related changes
- **ui**: User interface changes
- **components**: React component changes
- **utils**: Utility function changes
- **config**: Configuration changes
- **docs**: Documentation changes
- **tests**: Test-related changes

# Tagging Strategy

## Version Format

Following [Semantic Versioning](https://semver.org/) (<https://semver.org/>):

- **MAJOR.MINOR.PATCH** (e.g., 1.2.3)
- **MAJOR.MINOR.PATCH-PRERELEASE** (e.g., 2.0.0-beta.1)
- **MAJOR.MINOR.PATCH-PHASE** (e.g., 1.0.0-phase1)

## Tag Types

### Release Tags

```
v1.0.0      # Major release
v1.1.0      # Minor release
v1.1.1      # Patch release
v2.0.0-beta.1 # Pre-release
```

### Phase Tags

```
v1.0.0-phase1 # Phase 1 completion
v1.5.0-phase2 # Phase 2 completion
v2.0.0-phase3 # Phase 3 completion
```

### Milestone Tags

```
mvp-complete   # MVP milestone
beta-release   # Beta release milestone
production-ready # Production ready milestone
```

## Tagging Commands

```
# Create annotated tag
git tag -a v1.0.0 -m "Release version 1.0.0"

# Create phase tag
git tag -a v1.0.0-phase1 -m "Phase 1 MVP complete"



# Push tags
git push origin --tags

# List tags
git tag -l

# Delete tag
git tag -d v1.0.0
git push origin --delete v1.0.0
```

## Branch Protection Rules

### Main Branch Protection

-  Require pull request reviews before merging
-  Require status checks to pass before merging

- ☒ Require branches to be up to date before merging
- ☒ Require conversation resolution before merging
- ☒ Restrict pushes that create files larger than 100MB
- ☒ Do not allow bypassing the above settings

## Develop Branch Protection

- ☒ Require pull request reviews before merging
- ☒ Require status checks to pass before merging
- ☒ Require branches to be up to date before merging
- ⚠ Allow administrators to bypass pull request requirements

## Status Checks Required

- ☒ Continuous Integration (Tests)
- ☒ Code Quality (ESLint)
- ☒ Security Scan
- ☒ Build Success

## Remote Repository Setup

---

### GitHub Repository Configuration

#### 1. Create Repository

```
bash
# Create on GitHub, then:
git remote add origin https://github.com/username/berean-bible-app.git
git push -u origin main
git push origin develop --tags
```

#### 2. Configure Branch Protection

- Go to Settings → Branches
- Add protection rules for `main` and `develop`
- Configure required status checks

#### 3. Setup GitHub Actions

- Configure CI/CD workflows
- Set up automated testing
- Configure deployment pipelines

### Alternative: GitLab Setup

```
# GitLab setup
git remote add origin https://gitlab.com/username/berean-bible-app.git
git push -u origin main
git push origin develop --tags
```

## Collaboration Guidelines

---

### Pull Request Process

1. **PR Title:** Use conventional commit format
2. **Description:** Include purpose, changes, and testing notes

3. **Labels:** Apply appropriate labels (feature, bug, documentation)
4. **Reviewers:** Assign at least one reviewer
5. **Checks:** Ensure all automated checks pass
6. **Testing:** Include testing instructions

## Code Review Guidelines

1. **Review Scope:** Focus on logic, performance, security
2. **Feedback:** Provide constructive, specific feedback
3. **Approval:** Approve only when confident in changes
4. **Response Time:** Respond to reviews within 24 hours

## Conflict Resolution

1. **Prevention:** Regularly sync with base branch
2. **Resolution:** Use `git rebase` for clean history
3. **Communication:** Discuss complex conflicts with team
4. **Documentation:** Update relevant documentation

## Troubleshooting

---

### Common Issues

#### 1. Merge Conflicts

```
bash
git checkout feature-branch
git rebase develop
# Resolve conflicts
git add .
git rebase --continue
```

#### 2. Accidental Commits to Wrong Branch

```
bash
git log --oneline -n 5 # Find commit hash
git checkout correct-branch
git cherry-pick <commit-hash>
git checkout wrong-branch
git reset --hard HEAD~1
```

#### 3. Force Push After Rebase

```
bash
git push --force-with-lease origin feature-branch
```

## Recovery Procedures

1. **Lost Commits:** Use `git reflog` to find lost commits
  2. **Corrupted Branch:** Recreate from known good state
  3. **Wrong Merge:** Use `git revert` for public branches
-



This branching strategy ensures clean, maintainable code history while supporting parallel development across multiple phases and features. For questions or clarifications, contact the development team or create an issue in the repository.