# Feature Branch Workflow Guide

## Branch Structure

```
main (production)
  ↑
dev (development/integration)
  ↑
feature/* (individual features)
```

## Initial Setup

```
# Clone repository
git clone https://github.com/omanandswami2005/DMOR-OMS-Internship.git
cd DMOR-OMS-Internship

# Verify branches
git branch -a

# Switch to dev branch
git checkout dev
git pull origin dev
```

## Feature Development Flow

### 1. Create Feature Branch

```
# Always start from dev
git checkout dev
git pull origin dev

# Create feature branch
git checkout -b feature/invoice-management
```

## 2. Development Work

```
# Make changes
# ... code your feature ...

# Check status
git status

# Stage changes
git add .

# Commit (triggers pre-commit hooks: ESLint + Prettier)
git commit -m "feat: add invoice creation API

- Create invoice module in server/src/modules/invoice
- Add InvoiceService, InvoiceRepository, InvoiceController
- Implement invoice routes with validation
- Add invoice types and DTOs"
```

## 3. Push Feature Branch

```
# Push to remote
git push origin feature/invoice-management

# Or set upstream
git push -u origin feature/invoice-management
```

## 4. Keep Feature Branch Updated

```
# While working, sync with dev regularly
git checkout dev
git pull origin dev

git checkout feature/invoice-management
git merge dev

# Resolve conflicts if any
# ... fix conflicts ...
git add .
git commit -m "merge: sync with dev branch"
git push origin feature/invoice-management
```

## 5. Complete Feature

```
# Final sync before merge
git checkout dev
git pull origin dev

git checkout feature/invoice-management
git merge dev

# Push final version
git push origin feature/invoice-management
```

## 6. Merge to Dev

```
# Switch to dev
git checkout dev
git pull origin dev

# Merge feature branch
git merge feature/invoice-management

# Run tests
cd client && npm run build
cd ../server && npm start

# Push to dev
git push origin dev

# Delete feature branch (optional)
git branch -d feature/invoice-management
git push origin --delete feature/invoice-management
```

## 7. Release to Main (Production)

```
# When dev is stable and ready for production
git checkout main
git pull origin main

# Merge from dev
git merge dev

# Tag release
git tag -a v1.2.0 -m "Release v1.2.0: Invoice Management"

# Push to main with tags
git push origin main --tags
```

## Example: Multiple Developers Working

### Developer A: Invoice Feature

```
# Developer A
git checkout dev
git pull origin dev
git checkout -b feature/invoice-management

# ... work on invoice module ...
git add .
git commit -m "feat: add invoice CRUD operations"
git push origin feature/invoice-management
```

### Developer B: Reports Feature

```
# Developer B
git checkout dev
git pull origin dev
git checkout -b feature/sales-reports

# ... work on reports module ...
git add .
git commit -m "feat: add sales reports dashboard"
git push origin feature/sales-reports
```

### Developer C: Fix Bug in Dev

```
# Developer C fixes bug found in dev
git checkout dev
git pull origin dev
git checkout -b bugfix/order-calculation

# ... fix bug ...
git add .
git commit -m "fix: correct order total calculation"
git push origin bugfix/order-calculation

# Merge to dev immediately
git checkout dev
git merge bugfix/order-calculation
git push origin dev
```

### Developer A: Sync with Bug Fix

```
# Developer A syncs feature branch with bug fix
git checkout dev
git pull origin dev

git checkout feature/invoice-management
git merge dev
git push origin feature/invoice-management
```

## Conflict Resolution Example

```
# When merging dev into feature branch
git checkout feature/invoice-management
git merge dev

# If conflicts occur:
# CONFLICT (content): Merge conflict in client/src/router/routes.tsx

# Open conflicted file
code client/src/router/routes.tsx

# Resolve conflict markers:
<<<<<<< HEAD
import InvoicePage from '@/features/invoice/pages/InvoicePage';
```

```
=======
import ReportsPage from '@/features/reports/pages/ReportsPage';
>>>>>>> dev

# After resolving:
import InvoicePage from '@/features/invoice/pages/InvoicePage';
import ReportsPage from '@/features/reports/pages/ReportsPage';

# Stage resolved files
git add client/src/router/routes.tsx
git commit -m "merge: resolve conflicts with dev"
git push origin feature/invoice-management
```

## Complete Workflow Example

```
# Day 1: Start new feature
git checkout dev
git pull origin dev
git checkout -b feature/dispatch-tracking
# ... work ...
git add .
git commit -m "feat: add dispatch tracking module structure"
git push -u origin feature/dispatch-tracking

# Day 2: Continue work + sync
git checkout dev
git pull origin dev
git checkout feature/dispatch-tracking
git merge dev
# ... more work ...
git add .
git commit -m "feat: implement dispatch status updates"
git push origin feature/dispatch-tracking

# Day 3: Final work
# ... complete feature ...
git add .
git commit -m "feat: add dispatch reports and notifications"
git push origin feature/dispatch-tracking

# Merge to dev
git checkout dev
git pull origin dev
git merge feature/dispatch-tracking
npm run test
git push origin dev
git branch -d feature/dispatch-tracking

# Week later: Release to production
git checkout main
git pull origin main
git merge dev
git tag -a v1.3.0 -m "Release v1.3.0: Dispatch Tracking"
git push origin main --tags
```

## Branch Naming Conventions

```
feature/invoice-management        # New feature
feature/user-authentication       # New feature
bugfix/order-total-calculation    # Bug fix
hotfix/security-vulnerability     # Critical fix for production
refactor/cleanup-types            # Code refactoring
docs/api-documentation            # Documentation updates
```

## Commit Message Conventions

```
feat: add new feature
fix: bug fix
refactor: code refactoring
docs: documentation changes
style: formatting, missing semicolons, etc.
test: adding tests
chore: maintenance tasks
perf: performance improvements
```

## Quick Reference Commands

```
# Start new feature
git checkout dev && git pull && git checkout -b feature/name
```

```
# Daily sync
git checkout dev && git pull && git checkout - && git merge dev

# Complete feature
git checkout dev && git merge feature/name && git push

# Emergency hotfix
git checkout main && git checkout -b hotfix/name
# ... fix ...
git checkout main && git merge hotfix/name && git push
git checkout dev && git merge hotfix/name && git push

# View branches
git branch -a

# View commit history
git log --oneline --graph --all

# Undo last commit (keep changes)
git reset --soft HEAD~1

# Stash changes temporarily
git stash
git stash pop
```

## Team Workflow

### Sprint Planning

```
# Team decides features for sprint
# Each developer gets assigned features

# Developer 1: Invoice
git checkout -b feature/invoice-management

# Developer 2: Inventory
git checkout -b feature/inventory-alerts

# Developer 3: Reports
git checkout -b feature/monthly-reports
```

### Daily Standups

```
# Each developer syncs and reports progress
git checkout dev && git pull
git checkout feature/my-feature && git merge dev
git push origin feature/my-feature
```

### End of Sprint

```
# All features merged to dev
git checkout dev
git merge feature/invoice-management
git merge feature/inventory-alerts
git merge feature/monthly-reports

# Test dev branch
npm run test
npm run build

# Deploy to staging environment
# ... test staging ...

# Release to production
git checkout main
git merge dev
git tag -a v2.0.0 -m "Sprint 5 Release"
git push origin main --tags
```

## Pre-commit Hooks

Automatic checks before every commit:

```
# On commit, the following runs automatically:
✓ Running ESLint on staged files
✓ Running Prettier on staged files
✓ Formatting code

# If errors found:
✖ ESLint found errors - fix before committing
```

## CI/CD Integration

```
# .github/workflows/ci.yml (example)
on:
  push:
    branches: [dev, main]
  pull_request:
    branches: [dev, main]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - run: pnpm install
      - run: pnpm lint
      - run: pnpm test
      - run: pnpm build
```

## Best Practices

```
# ✅ DO
git checkout dev && git pull            # Always sync before starting
git commit -m "feat: clear message"     # Use conventional commits
git merge dev                           # Sync feature regularly
npm run build                           # Test before merging

# ❌ DON'T
git commit -m "fix stuff"               # Vague messages
git push --force                        # Never force push to dev/main
# work on dev directly                  # Always use feature branches
git merge main                          # Don't merge main to feature
```