Git Team Workflow Cheat Sheet

For 4-Person Teams

Branching Strategy

```
main (production-ready, always deployable)
  ↓
develop (integration branch, where features come together)
  ↓
feature/auth-jwt (individual feature branches)
feature/game-pong
feature/chat-websocket
fix/login-bug
```

Key Principle: Never commit directly to `main` or `develop`

Branch Naming Convention

Prefix	Purpose	Example
`feature/`	New features	`feature/oauth-login`
`fix/`	Bug fixes	`fix/database-connection`
`refactor/`	Code improvements	`refactor/user-service`
`docs/`	Documentation	`docs/api-endpoints`

Daily Workflow

1. Starting a New Feature

```bash
```

```
# Get latest code
git checkout develop
git pull origin develop

# Create your feature branch
git checkout -b feature/your-feature-name
```

2. Working on Your Feature

```
bash

# Make changes to your code

# Stage and commit
git add .
git commit -m "feat(auth): Add JWT authentication"

# Push to remote
git push origin feature/your-feature-name
```

3. Merging Your Work

```
bash

# Push your branch to remote
git push origin feature/your-feature-name

# Create Pull Request (PR) on GitHub/GitLab
# Target: feature/your-feature → develop
# Wait for team review
# After approval → Merge → Delete feature branch
```

4. Keeping Your Branch Updated

```
bash
```

```
# Update your feature branch with latest develop
git checkout feature/your-feature
git pull origin develop

# If conflicts occur:
# 1. Resolve conflicts in your code editor
# 2. Stage resolved files
git add .
git commit -m "merge: Resolve conflicts with develop"
git push origin feature/your-feature
```

Golden Rules

1. ✅ **Always pull before starting work**

```bash
git pull origin develop
```

2. ❌ **Never force push to shared branches** (main, develop)

```bash
# DON'T DO THIS on main or develop
git push --force
```

3. ✅ **Use Pull Requests (PRs)** - let teammates review your code
4. ✅ **Small, frequent commits** are better than one giant commit
5. ✅ **Keep `main` always deployable** - only merge stable code
6. ✅ **Delete feature branches** after merging to keep repo clean
7. ✅ **Communicate with your team** about what you're working on

---

Common Commands

```bash
```

```
# See all branches
git branch -a

# Switch branches
git checkout branch-name

# Check current status
git status

# See commit history
git log --oneline

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

# Discard all local changes
git reset --hard

# Update local branch list
git fetch --prune
```

Merge Flow

```
Your work flow:
feature/your-feature → develop (via PR) → main (when stable)

Never:
feature/your-feature → main (skipping develop)
```

When Something Goes Wrong

Accidentally committed to wrong branch

```
bash
```

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

# Switch to correct branch
git checkout correct-branch

# Commit again
git add .
git commit -m "your message"
```

Need to update your PR after review

bash

```
# Make the requested changes
git add .
git commit -m "fix: Address PR review comments"
git push origin feature/your-feature
# PR automatically updates!
```

Merge conflicts

bash

```
# Pull latest develop
git pull origin develop

# Git will mark conflicts in files like:
# <<<<<<< HEAD
# your changes
# =======
# their changes
# >>>>>>> develop

# 1. Open conflicted files
# 2. Choose which code to keep
# 3. Remove conflict markers
# 4. Save files

git add .
git commit -m "merge: Resolve conflicts with develop"
git push origin feature/your-feature
```

Pro Tips

- Commit often with meaningful messages
- Pull from `develop` daily to avoid large conflicts
- Review your teammates' PRs - learn from their code
- Use `git status` frequently to know what's happening
- Test your code before pushing
- Don't leave branches open for weeks - merge or close them