PAROL6 Team Workflow Guide

GitHub Projects & Git Branching Strategy

Version: 2.0.0
Last Updated: 2026-01-14
For: PAROL6 Development Team

📋 Table of Contents

---

1. Introduction

This guide explains how our team collaborates on the PAROL6 project using:

- **GitHub Projects** for task management
- **Git branching** for organized development
- **Issue templates** for consistent tracking
- **Pull Requests** for code review

Goal: Professional, organized, thesis-worthy development workflow.

---

2. GitHub Projects Overview

What is GitHub Projects?

GitHub Projects is a built-in project management tool that helps teams:

- ✅ Track tasks visually (Kanban board)
- ✅ Link work to code (issues → commits → PRs)
- ✅ Automate workflows (auto-move cards)
- ✅ Generate progress reports

Why we use it:

- **Free** and integrated with our repository
- **Visible progress** for thesis documentation
- **Team coordination** without extra tools
- **Automatic updates** from git activity

Our Project Board Structure

```
┌──────────────────────────────────────────────────────────┐
│  PAROL6 ros2_control Migration                           │
├──────────────┬──────────────┬──────────────┬─────────────┤
│ 📋 Backlog   │ 🔄 In Prog   │ ✅ Done      │ 🐛 Blocked  │
├──────────────┼──────────────┼──────────────┼─────────────┤
│ Future work  │ Active tasks │ Complete     │ Issues      │
│ Planned      │ Assigned     │ Verified     │ Waiting     │
└──────────────┴──────────────┴──────────────┴─────────────┘
```

Column Meanings:

- **Backlog:** Tasks ready to be picked up
- **In Progress:** Currently being worked on
- **Done:** Completed and verified
- **Blocked:** Waiting on dependencies

---

3. Initial Setup

3. Initial Setup

3.1 Creating the Project Board

Step 1: Create the Project

1. Go to: https://github.com/Abdulkareem771/PAROL6_URDF
2. Click **"Projects"** tab
3. Click **"New project"**
4. Choose **"Board"** template
5. Name: **"PAROL6 ros2_control Migration"**
6. Click **"Create"**

Step 2: Customize Columns

1. Rename columns to:
   - 📋 `Backlog`
   - 🔄 `In Progress`
   - ✅ `Done`
2. Add new column: 🐛 `Blocked`

Step 3: Enable Automation

1. Click **"..."** (top right) → **"Workflows"**
2. Enable:
    - ✅ Auto-add new issues
    - ✅ Move to "In Progress" when assigned
    - ✅ Move to "Done" when closed

Step 4: Create Phase Issues

1. Go to **"Issues"** tab
2. Create issues from templates (Day 1-5)

---

4. Git Branching Strategy

4.1 Branch Structure

We follow **Git Flow** with simplified naming:

```
main (v2.0.0) ← Production-ready, tagged releases
│
├─ day2-serial-tx ← Feature branches for major work
├─ day3-feedback
├─ day4-first-motion
│
├─ bugfix/controller-crash ← Bug fixes
├─ docs/update-readme ← Documentation updates
└─ feature/velocity-control ← New features
```

4.2 Branch Naming Convention

Type	Format	Example
Day phases	`day[N]-[description]`	`day2-serial-tx`
Features	`feature/[name]`	`feature/velocity-control`
Bug fixes	`bugfix/[issue]`	`bugfix/controller-crash`
Documentation	`docs/[topic]`	`docs/update-readme`
Experiments	`experiment/[idea]`	`experiment/new-protocol`

4.3 Protected Branches

`main` branch rules:

- ✅ Only merge via Pull Request
- ✅ Require 1 approval (for team > 2 people)
- ✅ Must pass tests (if CI/CD setup)
- ❌ No direct commits

---

5. Daily Workflow

5.1 Starting Your Day

Morning Routine:

```
# 1. Update your local repository
cd ~/Desktop/PAROL6_URDF
git checkout main
git pull origin main

# 2. Check what you're working on
git branch  # See your branches

# 3. View project board
# Go to GitHub → Projects → Check your assignments
```

5.2 Starting a New Task

Step 1: Pick an Issue

1. Go to project board
2. Find unassigned issue in **Backlog**
3. Assign to yourself (right sidebar)
4. Move to **In Progress**

Step 2: Create a Branch

```
# From main (always start fresh from main!)
git checkout main
git pull origin main

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

# Example for Day 2:
git checkout -b day2-serial-tx
```

Step 3: Start Coding

```
# Make changes
# Test your changes
# Commit frequently

git add .
git commit -m "feat: implement serial port opening

- Add serial library include
```

```
- Implement on_configure()
- Add error handling

Relates to #2"  # Links to issue #2
```

5.3 During Development

Commit Message Format:

```
<type>: <short description>

<detailed explanation>

<issue reference>
```

Types:

- `feat:` New feature
- `fix:` Bug fix
- `docs:` Documentation
- `refactor:` Code restructuring
- `test:` Adding tests
- `chore:` Maintenance

Examples:

```
git commit -m "feat: add write() method for serial TX"
git commit -m "fix: resolve controller crash on startup"
git commit -m "docs: update Day 2 plan with timing data"
git commit -m "test: add unit tests for serial communication"
```

5.4 Pushing Your Work

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

# First time pushing a new branch:
git push -u origin feature/your-feature-name
```

5.5 End of Day

```
# Commit all work (even if incomplete)
git add .
git commit -m "wip: partial implementation of feature X"
```

```
git push

# Update issue with progress comment
# Go to GitHub → Your issue → Add comment about progress
```

---

6. Working with Issues

6.1 Creating an Issue

From Template:

1. Go to **Issues → New issue**
2. Choose template (Day 1-5, Bug Report)
3. Template auto-fills with checklist
4. Edit title if needed
5. Click **"Submit new issue"**

Custom Issue:

```
Title: [BUG] Controller fails to activate

Description:
**Problem:** Controller manager crashes when...
**Expected:** Should activate normally
**Environment:** Docker, ROS Humble
**Steps to reproduce:**
1. ...
2. ...
```

6.2 Linking Issues to Code

In Commit Messages:

```
git commit -m "fix: resolve crash - fixes #5"
git commit -m "feat: add feature - relates to #3"
git commit -m "docs: update guide - closes #7"
```

Keywords that auto-close issues:

- `fixes #N`
- `closes #N`
- `resolves #N`

6.3 Updating Issues

Add Progress Updates:

```
### Progress Update - 2026-01-14

- ✅ Completed serial port opening
- ✅ Tested with ESP32
- 🔄 Working on write() method
- ❓ Question: Should we use CRC16?
```

Mark Tasks Complete: Edit the issue description to check boxes:

```
- [x] Completed task
- [ ] Incomplete task
```

---

7. Creating Pull Requests

7.1 When to Create a PR

Create PR when:

- ✅ Feature is complete and tested
- ✅ Code follows style guidelines
- ✅ Documentation is updated
- ✅ Ready for team review

Don't wait too long! Small, frequent PRs are better than huge ones.

7.2 Creating the PR

Step 1: Push Your Branch

```
git push origin your-branch-name
```

Step 2: On GitHub

1. Go to repository
2. Click **"Pull requests"** tab
3. Click **"New pull request"**
4. **Base:** `main` ← **Compare:** `your-branch`
5. Click **"Create pull request"**

Step 3: Fill PR Template

```
Title: Add serial TX implementation (Day 2)

## Description
```

```
Implements serial transmission to ESP32 at 25Hz.

## Changes
- Implemented on_configure() with serial port opening
- Added write() method with command formatting
- Included timing guards (< 5ms)

## Testing
- ✅ Tested with ESP32 echo firmware
- ✅ Validated 25Hz rate, jitter < 1ms
- ✅ 15-minute stability test passed

## Closes
Closes #2

## Screenshots
(Optional - add terminal output or plots)
```

Step 4: Request Review

- On right sidebar, click **"Reviewers"**
- Select teammate
- They'll get notified

7.3 After Creating PR

PR checks:

1. **GitHub Actions** may run (if configured)
2. **Reviewer** will comment and approve
3. **Conflicts?** Resolve them (see section 9.3)

---

8. Code Review Process

8.1 As a Reviewer

Your Responsibilities:

1. ✅ Check code quality
2. ✅ Test the changes locally
3. ✅ Provide constructive feedback
4. ✅ Approve or request changes

How to Review:

1. Go to **Pull requests** tab
2. Click on PR to review
3. Click **"Files changed"** tab
4. Add comments:
    - **Line comment:** Click **"+"** next to line

- **General comment:** Use "Review changes"

Review Checklist:

```
- [ ] Code compiles without errors
- [ ] Tests pass
- [ ] Documentation updated
- [ ] No merge conflicts
- [ ] Follows coding standards
- [ ] Issue is actually resolved
```

Feedback Examples:

```
✅ Good: "Consider adding error handling here for robustness"
✅ Good: "Nice implementation! Small typo on line 45"
❌ Bad: "This is wrong"
❌ Bad: "I don't like this"
```

8.2 As the PR Author

Responding to Feedback:

1. Read all comments carefully
2. Make requested changes:

```
# Make changes
git add .
git commit -m "refactor: address review feedback"
git push
```

3. Reply to comments:
   - "✅ Fixed in commit abc123"
   - "❓ Can you clarify what you mean?"
   - "💡 Good idea, implemented!"

When Approved:

1. Reviewer clicks **"Approve"**
2. You click **"Merge pull request"**
3. Choose: **"Squash and merge"** or **"Merge commit"**
4. Delete branch after merge

---

9. Common Scenarios

9.1 Updating Your Branch with Latest Main

Scenario: Someone merged to `main`, you need those changes.

```
# Save your work first
git add .
git commit -m "wip: save current work"

# Update main
git checkout main
git pull origin main

# Go back to your branch
git checkout your-branch-name

# Merge main into your branch
git merge main

# If conflicts, resolve them (see 9.3)
# Then push
git push origin your-branch-name
```

9.2 Switching Between Tasks

```
# You're on feature-A, need to quickly fix bug

# Save current work
git add .
git commit -m "wip: partial feature A"
git push

# Switch to main and create bug fix branch
git checkout main
git checkout -b bugfix/critical-issue

# Fix bug
# ...
git commit -m "fix: critical bug"
git push

# Create PR and merge

# Go back to feature-A
git checkout feature-A
git merge main  # Get the bug fix
```

9.3 Resolving Merge Conflicts

When you see:

```
CONFLICT (content): Merge conflict in file.txt
```

Steps:

1. **Open conflicted file:**

```
<<<<<<< HEAD
Your changes
=======
Their changes
>>>>>>> main
```

2. **Decide what to keep:**
   - Keep yours: Delete `<<<`, `===`, `>>>` and their version
   - Keep theirs: Delete yours
   - Keep both: Merge manually

3. **Mark as resolved:**

```
git add file.txt
git commit -m "merge: resolve conflicts with main"
git push
```

9.4 Undoing a Commit

Before pushing:

```
# Undo last commit, keep changes
git reset --soft HEAD~1

# Undo last commit, discard changes (⚠️ DANGEROUS)
git reset --hard HEAD~1
```

After pushing:

```
# Create a reverse commit
git revert HEAD
git push
```

---

10. Troubleshooting

Issue: "Branch is behind main"

```
git checkout your-branch
git merge main
git push
```

Issue: "Permission denied"

```
# Check your git credentials
git config user.name
git config user.email

# Re-authenticate
git credential-osxkeychain erase  # Mac
git credential-cache exit  # Linux
```

Issue: "Cannot push to main"

Good! `main` is protected. Create a PR instead.

Issue: "Lost my changes"

```
# Check reflog (git history of everything)
git reflog

# Find your commit
git checkout <commit-hash>

# Create branch from it
git checkout -b recovery-branch
```

---

11. Best Practices

11.1 Commit Hygiene

✅ DO:

- Commit frequently (every few hours)
- Write clear commit messages
- Test before committing
- Keep commits focused on one change

❌ DON'T:

- Commit broken code to `main`

- Write vague messages like "fix stuff"
- Commit large binary files
- Mix unrelated changes in one commit

11.2 Branch Hygiene

✅ **DO:**

- Create branches from updated `main`
- Delete branches after merge
- Keep branches short-lived (< 1 week)
- Use descriptive branch names

❌ **DON'T:**

- Work directly on `main`
- Leave stale branches
- Create nested feature branches
- Reuse branch names

11.3 Communication

✅ **DO:**

- Comment on issues with progress
- Request help when blocked
- Tag teammates in discussions (@username)
- Update PR with changes

❌ **DON'T:**

- Ghost the team (push without updates)
- Ignore review feedback
- Leave issues unassigned
- Merge without approval

11.4 Code Quality

✅ **DO:**

- Follow project coding standards
- Add comments for complex logic
- Update documentation
- Write meaningful variable names

❌ **DON'T:**

- Rush to merge
- Skip testing
- Leave TODOs in production code
- Ignore compiler warnings

12. Quick Reference

Essential Git Commands

```
# Setup
git clone <url>                    # Clone repository
git config user.name "Your Name" # Set your name

# Daily workflow
git checkout main                  # Go to main
git pull origin main               # Update main
git checkout -b feature/name       # Create branch
git add .                          # Stage changes
git commit -m "message"            # Commit
git push origin branch-name        # Push

# Syncing
git fetch origin                   # Check for updates
git merge main                     # Merge main into your branch
git status                         # Check status
git log --oneline                  # View history

# Cleanup
git branch -d branch-name          # Delete local branch
git push origin --delete branch  # Delete remote branch
```

Issue Keywords

Keyword	Effect
`fixes #N`	Closes issue #N when PR merges
`closes #N`	Closes issue #N when PR merges
`resolves #N`	Closes issue #N when PR merges
`relates to #N`	Links to issue #N (doesn't close)
`refs #N`	Links to issue #N (doesn't close)

GitHub Shortcuts

Key	Action
`g i`	Go to Issues
`g p`	Go to Pull Requests
`g b`	Go to Projects
`t`	Search files

Key	Action
?	Show keyboard shortcuts

---

13. Getting Help

Team Communication

- **Slack/Discord:** Daily questions
- **GitHub Issues:** Feature requests, bugs
- **PR Comments:** Code-specific questions
- **Email:** Urgent matters

Resources

- **Repository:** https://github.com/Abdulkareem771/PAROL6_URDF
- **GitHub Docs:** https://docs.github.com
- **Git Tutorial:** https://git-scm.com/book/en/v2

Escalation

1. **Try to solve yourself** (15 min)
2. **Ask teammate** (GitHub comment)
3. **Create issue** (if it's a blocker)
4. **Team meeting** (if design decision needed)

---

14. Appendix: Project Milestones

Current Status: v2.0.0

Phase	Status	Target Date
Day 1: SIL	✅ Complete	2026-01-14
Day 2: Serial TX	🔄 In Progress	2026-01-17
Day 3: Feedback	📋 Planned	2026-01-20
Day 4: First Motion	📋 Planned	2026-01-22
Day 5: Validation	📋 Planned	2026-01-24

Release History

- **v2.0.0** (2026-01-14) - Day 1 SIL validation complete
- **v1.1.0** (Planned) - Serial TX
- **v1.2.0** (Planned) - Feedback loop
- **v2.0.0** (Planned) - Hardware deployment

Document Version: 1.0
Last Updated: 2026-01-14
Maintained by: PAROL6 Team Lead

Converting to PDF

Using Pandoc (Recommended):

```
pandoc TEAM_WORKFLOW_GUIDE.md -o TEAM_WORKFLOW_GUIDE.pdf \
  --pdf-engine=xelatex \
  -V geometry:margin=1in \
  -V colorlinks=true \
  --toc
```

Using VS Code:

1. Install "Markdown PDF" extension
2. Right-click file → "Markdown PDF: Export (pdf)"

Using Online Tools:

- https://www.markdowntopdf.com/
- https://md2pdf.netlify.app/