# workflow_mini

July 22, 2025

## ZZCOLLAB Mini Workflow Guide

### Solo Developer: Complete Analysis Workspace

#### Single Developer Setup (Complete Environment)

For solo developers who want a complete, reproducible analysis environment:

```
# Option A: Two-step approach (recommended for flexibility)

# Step 1: Create team images (builds all variants: shell, rstudio, verse)
zzcollab -i -t rgt47 -p c275 -B all -S

# Step 2: Create full project with personal workspace
mkdir c275 && cd c275
zzcollab -t rgt47 -p c275 -F -d ~/dotfiles --github -I shell
```

**What this creates:** - [CHECKMARK] **Complete Docker environment**: All variants (shell, rstudio, verse) available - [CHECKMARK] **Personal workspace**: Your dotfiles integrated - [CHECKMARK] **Private GitHub repository**: `https://github.com/rgt47/c275` with CI/CD - [CHECKMARK] **Ready-to-code**: Start immediately with `make docker-zsh`

#### Alternative: One-command setup

```
# Single command for complete solo setup
zzcollab -t rgt47 -p c275 -F -d ~/dotfiles --github -I shell
```

**Build modes:** `-F` (Fast), `-S` (Standard), `-C` (Comprehensive)
**Interfaces:** `-I shell` (vim/tmux), `-I rstudio` (web), `-I verse` (publishing)

### Daily Workflow

```
make docker-zsh          # Start development
# ... analysis work ...
exit                     # Exit container
git add . && git commit -m "Add analysis" && git push
```

---

## Developer 1: Team Lead Project Initialization

### Two-Step Process for Team Lead (Fixed -i Flag Behavior)

```
# Step 1: Create and push team Docker images ONLY
# Navigate to projects directory first
cd ~/projects  # or your preferred projects directory

# Create team images with selective base image building
# Choose one approach based on team needs:

# Option A: Build only shell variant (fastest - recommended for command-line teams)
zzcollab -i -t rgt47 -p png1 -B r-ver -S

# Option B: Build only RStudio variant (for GUI-focused teams)
zzcollab -i -t rgt47 -p png1 -B rstudio -S

# Option C: Build only verse variant (for publishing-focused teams)
zzcollab -i -t rgt47 -p png1 -B verse -S

# Option D: Build all variants (traditional approach - takes longer)
zzcollab -i -t rgt47 -p png1 -B all -S

# Step 2: Create full project structure (run separately)
mkdir png1 && cd png1  # or git clone if repo exists
zzcollab -t rgt47 -p png1 -I shell  # Full project setup (use appropriate interface)

# Note: Step 1 (-i flag) now stops after team image creation
# Step 2 creates the actual project structure and GitHub repository
```

**What Each Step Does:**

**Step 1 (-i flag):** 1. **Builds team Docker images**: Creates and pushes to Docker Hub as `rgt47/png1core-shell:latest` (and/or other variants) 2. **Stops after image creation**: Provides guidance for next steps

**Step 2 (separate project setup):** 1. **Creates project directory**: `png1/` 2. **Initializes zzcollab project structure**: Complete R package with analysis framework 3. **Creates private GitHub repository**: `https://github.com/rgt47/png1` 4. **Sets up automated CI/CD**: GitHub Actions for team image management 5. **Enables team collaboration**: Dev 2 and Dev 3 can join immediately

**Required: Invite Team Members**

After completing both steps, Dev 1 must invite collaborators:

```
# Invite team members to the private GitHub repository
gh repo invite rgt47/png1 dev2-github-username
gh repo invite rgt47/png1 dev3-github-username

# Alternative: Via GitHub web interface
# 1. Go to https://github.com/rgt47/png1/settings/access
# 2. Click "Invite a collaborator"
# 3. Add dev2-github-username and dev3-github-username with "Write" access
```

**For Dev 2 and Dev 3 to Join:**

```
# 1. Accept GitHub collaboration invitation
# Check email for invitation from rgt47/png1 repository
# OR visit: https://github.com/rgt47/png1/invitations
# Click "Accept invitation"

# 2. Clone the project
git clone https://github.com/rgt47/png1.git
cd png1

# 3. Join with available interface (they'll get helpful errors if variant unavailable)
zzcollab -t rgt47 -p png1 -I shell -d ~/dotfiles      # If shell variant available
zzcollab -t rgt47 -p png1 -I rstudio -d ~/dotfiles    # If RStudio variant available
zzcollab -t rgt47 -p png1 -I verse -d ~/dotfiles      # If verse variant available

# 4. Start development
```

```
make docker-zsh        # or make docker-rstudio, make docker-verse
```

**Key Benefits of This Approach:**

- [CHECKMARK] **No local workspace for Dev 1**: Team infrastructure created without personal development setup
- [CHECKMARK] **Faster initialization**: Only builds needed variants, not all three
- [CHECKMARK] **Immediate team access**: Dev 2 & 3 can join as soon as GitHub repo is created
- [CHECKMARK] **Flexible team scaling**: Can add more variants later with `zzcollab -V rstudio`
- [CHECKMARK] **Error guidance**: Team members get helpful messages if requesting unavailable variants

**If Team Needs Multiple Interfaces Later:**

Developer 1 can add variants incrementally:

```
cd png1
zzcollab -V rstudio    # Add RStudio variant
zzcollab -V verse      # Add verse variant for publishing
```

This approach optimizes for **team coordination** while minimizing **setup overhead** for the team lead! [ROCKET]

---

## Developer 2: Development Completion Workflow

When **Developer 2** finishes their development work, here's the complete workflow:

### 1. Final Testing & Validation (Inside Container)

```
# Still in development container (make docker-zsh)
R
# Run final tests
devtools::load_all()           # Load all package functions
devtools::test()               # Run unit tests
testthat::test_dir("tests/integration")  # Run integration tests
source("scripts/my_analysis.R")  # Test your analysis script
quit()
```

### 2. Exit Container & Validate Dependencies

```
# Exit the development container
exit

# Validate all dependencies are properly tracked
make docker-check-renv-fix     # Auto-fix any dependency issues
make docker-test               # Run all tests in clean environment
make docker-render             # Ensure reports render correctly
```

### 3. Git Workflow - Commit Changes

```
# Check what you've changed
git status
git diff

# Stage and commit your work
git add .
git commit -m "Add [feature description] with comprehensive tests

- [Describe what you implemented]
- [List any new packages added]
- [Mention test coverage]
- All tests passing and dependencies validated"

# Push to your feature branch (if using feature branches - recommended)
git push origin feature/my-analysis

# OR push directly to main (if using simple workflow)
git push origin main
```

### 4. Create Pull Request (Recommended Team Workflow)

```
# Create PR for team review
gh pr create --title "Add [feature description]" \
    --body "## Summary
- [Describe your contribution]
- [List any new analysis scripts/functions]
- [Mention if new packages were added]

## Testing
```

```
- [x] All unit tests pass
- [x] Integration tests pass
- [x] Analysis scripts run without errors
- [x] Report renders successfully
- [x] Dependencies validated

## Impact
- [Describe how this affects the project]
- [Any breaking changes or requirements for other devs]"
```

### 5. What Happens Next (Automated)

When Dev 2 pushes changes:

1. **GitHub Actions automatically**:
   - [CHECKMARK] Runs R package validation
   - [CHECKMARK] Executes all tests
   - [CHECKMARK] Renders analysis reports
   - [CHECKMARK] **Detects if new packages were added**
2. **If new packages detected**:
   - [CHECKMARK] **Rebuilds team Docker image** with new packages
   - [CHECKMARK] **Pushes updated image** to Docker Hub (`rgt47/png1core-*:latest`)
   - [CHECKMARK] **Notifies team** via commit comment with update instructions
3. **Team gets notification**:

   ```
   [WHALE] Team Docker Image Updated

   New packages detected: tidymodels, plotly

   Team members: Update your environment with:
   git pull
   docker pull rgt47/png1core-shell:latest
   make docker-zsh
   ```

### 6. Team Synchronization (Dev 1 & Dev 3)

Other team members sync automatically:

```
# Dev 1 and Dev 3 run when they see the notification:
git pull                           # Get latest code changes
docker pull rgt47/png1core-shell:latest  # Get updated team environment
make docker-zsh                    # Continue development with new packages
```

### Alternative: Simple Direct Push Workflow

If not using pull requests:

```
# After validation (steps 1-2 above)
git add .
git commit -m "Add my analysis with tests - all dependencies validated"
git push origin main               # Direct push triggers team image rebuild
```

### Key Benefits of This Workflow:

- [CHECKMARK] **Zero manual image management**: GitHub Actions handles Docker rebuilds
- [CHECKMARK] **Automatic team notification**: Everyone knows when environment updates
- [CHECKMARK] **Dependency validation**: Prevents environment drift before commit
- [CHECKMARK] **Professional quality**: Tests, validation, and documentation required
- [CHECKMARK] **Team coordination**: Clear communication about changes and impacts

### Dev 2's Work is Done! [PARTY]

Once Dev 2 pushes their changes: - **Code is integrated** into the main project - **Team environment is updated** automatically
- **Other developers are notified** and can sync - **Dev 2 can start next feature** or analysis

This workflow ensures **zero-friction collaboration** while maintaining **enterprise-grade quality standards**! [ROCKET]

---

## Developer 1: Reacting to Team Contributions & Adding Own Work

Here are the commands **Developer 1 (Team Lead)** uses to react to Dev 2 and Dev 3's additions and then add their own code:

### 1. Sync with Team Changes

```
# Navigate to project directory
cd png1

# Get latest code changes from team
git pull origin main

# Get latest team environment (automatically updated by GitHub Actions)
docker pull rgt47/png1core-shell:latest   # or whatever variant you use

# Check what changed
git log --oneline -10                      # See recent commits
git diff HEAD~3                            # See changes since 3 commits ago
```

### 2. Review Team Contributions (Optional)

```
# Review specific team member changes
git log --author="dev2" --oneline -5       # See Dev 2's recent commits
git log --author="dev3" --oneline -5       # See Dev 3's recent commits

# Look at specific files that changed
git show HEAD~1                            # Show last commit details
git diff HEAD~2..HEAD scripts/             # See script changes
git diff HEAD~2..HEAD R/                   # See function changes
```

### 3. Start Development Environment with Updated Team Packages

```
# Enter updated development environment
make docker-zsh                           # All team packages now available

# Verify environment is up to date
R
installed.packages()[,1]                   # Check available packages
devtools::load_all()                      # Load all team functions
devtools::test()                          # Run all tests to ensure compatibility
quit()
```

### 4. Explore Team's New Code (Inside Container)

```
# Review what Dev 2 and Dev 3 added
ls scripts/                                 # See new analysis scripts
ls R/                                       # See new functions
ls tests/                                   # See new tests

# Test their analysis scripts
R
source("scripts/dev2_analysis.R")           # Run Dev 2's analysis
source("scripts/dev3_visualization.R")      # Run Dev 3's work
# Understand their approach and results
quit()
```

### 5. Create Feature Branch for Own Work

```
# Create branch for your new work
git checkout -b feature/dev1-integration

# OR work directly on main (simpler workflow)
# git checkout main
```

### 6. Add Your Own Code (Inside Container)

```
# Still in development container
vim scripts/04_advanced_modeling.R          # Create your analysis

# Example: Build on team's work
vim R/integration_functions.R               # Add functions that use team's work

# Write tests for your additions
vim tests/testthat/test-integration_functions.R
vim tests/integration/test-04_advanced_modeling.R

# Test your new code
R
devtools::load_all()                        # Load all functions (yours + team's)
source("scripts/04_advanced_modeling.R") # Test your script
devtools::test()                            # Run all tests
quit()
```

### 7. Exit Container & Validate Complete Integration

```
# Exit development container
exit

# Validate entire project works together
make docker-check-renv-fix                # Ensure dependencies are tracked
make docker-test                          # Run all tests (team's + yours)
make docker-render                        # Ensure reports still render

# Test end-to-end workflow
make docker-zsh
R
# Run complete analysis pipeline
source("scripts/01_data_import.R")        # Original work
source("scripts/dev2_analysis.R")         # Dev 2's contribution
source("scripts/dev3_visualization.R")    # Dev 3's contribution
source("scripts/04_advanced_modeling.R") # Your new integration
quit()
exit
```

### 8. Commit Your Integration Work

```
# Check what you've added
git status
git diff

# Commit your work
git add .
git commit -m "Add advanced modeling integration building on team contributions

- Integrate Dev 2's analysis patterns with advanced modeling
- Extend Dev 3's visualization framework for model results
- Add comprehensive integration tests for complete pipeline
- All team code compatibility maintained and tested"

# Push to feature branch
git push origin feature/dev1-integration

# OR push directly to main
# git push origin main
```

#### 9. Create Pull Request for Team Review

```
# Create PR for team feedback
gh pr create --title "Add advanced modeling integration" \
    --body "## Summary
- Built advanced modeling on top of Dev 2's analysis framework
- Extended Dev 3's visualization tools for model interpretation
- Added comprehensive integration testing

## Integration Testing
- [x] All existing team code runs without modification
- [x] New code integrates seamlessly with team contributions
- [x] Complete analysis pipeline tested end-to-end
- [x] All dependencies validated

## Team Impact
- Enhances existing analysis without breaking changes
- Provides advanced modeling capabilities for future work
- Maintains all existing functionality"
```

#### 10. Alternative: Quick Integration (Direct Push)

```
# For simple additions, skip PR process
git add .
git commit -m "Add modeling integration - builds on team's excellent foundation"
git push origin main                    # Triggers automatic team image rebuild
```

### Key Benefits of This Workflow:

- [CHECKMARK] **Seamless integration**: Dev 1 builds on team work without conflicts
- [CHECKMARK] **Automatic environment sync**: GitHub Actions handled package updates
- [CHECKMARK] **Code compatibility**: Testing ensures nothing breaks
- [CHECKMARK] **Team coordination**: PR process enables feedback and discussion
- [CHECKMARK] **Professional quality**: Integration testing validates entire pipeline

**What Happens Next:**

1. **GitHub Actions automatically**:
   - [CHECKMARK] Tests complete integration (all team code + Dev 1's additions)
   - [CHECKMARK] Rebuilds team image if new packages added
   - [CHECKMARK] Notifies team of updated environment
2. **Team members sync**:

```
git pull                                  # Get Dev 1's integration work
docker pull rgt47/png1core-shell:latest  # Get any env updates
make docker-zsh                           # Continue with enhanced codebase
```

This workflow ensures **Dev 1 can lead and integrate** while **building on the team's excellent contributions**! [ROCKET]