workflow_mini

August 04, 2025

ZZCOLLAB Mini Workflow Guide

Configuration System (Recommended)

ZZCOLLAB includes a configuration system to eliminate repetitive typing and set project defaults. **Set this up once and simplify all subsequent commands.**

One-Time Configuration Setup

```
# 1. Initialize configuration file
zzcollab --config init

# 2. Set your defaults (customize as needed)
zzcollab --config set team-name "rgt47"              # Your Docker Hub
                                                        # account
zzcollab --config set github-account "rgt47"        # Your GitHub
                                                        # username
zzcollab --config set build-mode "standard"         # fast, standard,
                                                        # comprehensive
zzcollab --config set dotfiles-dir "~/dotfiles"     # Path to your
                                                        # dotfiles

# 3. View your configuration
zzcollab --config list
```

Config-Aware Workflows

With configuration set up, commands become much simpler:

```
# Traditional verbose approach:
zzcollab -i -t rgt47 -p myproject -B rstudio -S -d ~/dotfiles
```

```
# Config-simplified approach (identical result):
zzcollab -i -p myproject -B rstudio
```

All workflows below show both approaches - use whichever you prefer!

Solo Developer: Complete Analysis Workspace

Prerequisites: Install ZZCOLLAB (One-time)

```
# 1. Clone and install zzcollab system
git clone https://github.com/rgt47/zzcollab.git
cd zzcollab && ./install.sh

# 2. Verify installation
zzcollab --help && which zzcollab

# 3. Optional: Set up configuration (recommended)
zzcollab --config init
zzcollab --config set team-name "rgt47"
zzcollab --config set build-mode "standard"
zzcollab --config set dotfiles-dir "~/dotfiles"
```

Single Developer Setup (Complete Environment)

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

```
# Option A: With Configuration (Recommended)
# Step 1: Create team images (builds all variants: shell, rstudio, verse)
zzcollab -i -p c275 -B all

# Step 2: Create full project with personal workspace
mkdir c275 && cd c275
zzcollab -p c275 -F --github -I shell

# Option B: Traditional Verbose Approach
# 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`

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

Prerequisites: Install ZZCOLLAB (One-time)

```
# 1. Clone and install zzcollab system
git clone https://github.com/rgt47/zzcollab.git
cd zzcollab && ./install.sh

# 2. Verify installation
zzcollab --help && which zzcollab

# 3. Optional: Set up configuration (recommended for team leads)
zzcollab --config init
zzcollab --config set team-name "rgt47"          # Your Docker Hub team name
zzcollab --config set github-account "rgt47"     # Your GitHub account
zzcollab --config set build-mode "standard"      # Default build mode
zzcollab --config set dotfiles-dir "~/dotfiles"  # Your dotfiles path
```

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:

# With Configuration (Recommended - much simpler):
zzcollab -i -p png1 -B r-ver          # Build only shell variant (fastest)
zzcollab -i -p png1 -B rstudio        # Build only RStudio variant
                                       # (GUI teams)
zzcollab -i -p png1 -B verse          # Build only verse variant
                                       # (publishing)
zzcollab -i -p png1 -B all            # Build all variants (traditional)

# Traditional Verbose Approach:
zzcollab -i -t rgt47 -p png1 -B r-ver -S     # Build only shell variant
zzcollab -i -t rgt47 -p png1 -B rstudio -S   # Build only RStudio variant
zzcollab -i -t rgt47 -p png1 -B verse -S     # Build only verse variant
zzcollab -i -t rgt47 -p png1 -B all -S       # Build all variants

# Step 2: Create full project structure (run separately)
mkdir png1 && cd png1  # or git clone if repo exists

# With Configuration:
zzcollab -p png1 -I shell             # Full project setup with shell
                                       # interface

# Traditional Verbose:
zzcollab -t rgt47 -p png1 -I shell    # Full project setup with shell
                                       # 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:

Prerequisites: Install ZZCOLLAB (One-time)

```
# 0. Clone and install zzcollab system
git clone https://github.com/rgt47/zzcollab.git
cd zzcollab && ./install.sh && zzcollab --help

# 1. Optional: Set up configuration for easier commands
zzcollab --config init
zzcollab --config set team-name "rgt47"         # Match team settings
zzcollab --config set dotfiles-dir "~/dotfiles" # Your dotfiles path

# 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 (helpful errors if variant unavailable)
# With Configuration (Recommended):
zzcollab -p png1 -I shell      # If shell available
zzcollab -p png1 -I rstudio    # If RStudio available
zzcollab -p png1 -I verse      # If verse available

# Traditional Verbose Approach:
zzcollab -t rgt47 -p png1 -I shell -d ~/dotfiles    # If shell available
zzcollab -t rgt47 -p png1 -I rstudio -d ~/dotfiles  # If RStudio available
zzcollab -t rgt47 -p png1 -I verse -d ~/dotfiles    # If verse 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

Prerequisites: Install ZZCOLLAB (One-time)

```
# 1. Clone and install zzcollab system
git clone https://github.com/rgt47/zzcollab.git
cd zzcollab && ./install.sh && zzcollab --help

# 2. Optional: Set up configuration for simplified commands
zzcollab --config init
zzcollab --config set team-name "rgt47"          # Team name for this
                                                 # project
zzcollab --config set build-mode "fast"          # Your preferred mode
zzcollab --config set dotfiles-dir "~/dotfiles"  # Your dotfiles path
```

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

Prerequisites: Install ZZCOLLAB (One-time)

```
# 1. Clone and install zzcollab system
git clone https://github.com/rgt47/zzcollab.git
cd zzcollab && ./install.sh && zzcollab --help

# 2. Configuration should already be set up from team initialization
# If not, set it up:
zzcollab --config init
zzcollab --config set team-name "rgt47"
zzcollab --config set dotfiles-dir "~/dotfiles"
```

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 tests for 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 using
                                               # 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
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 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]

Developer 2: Ubuntu Setup - Fresh Lenovo ThinkPad

New Developer Environment Setup (Ubuntu)

When **Developer 2** gets a brand new Lenovo ThinkPad with fresh Ubuntu installation, here are all the required setup steps to join the team analysis:

Prerequisites: System Setup (One-time Ubuntu Installation)

```
# 1. Update system packages
sudo apt update && sudo apt upgrade -y

# 2. Install essential development tools
sudo apt install -y \
    git \
    curl \
    wget \
    build-essential \
    ca-certificates \
    gnupg \
    lsb-release

# 3. Install Docker Engine (official Ubuntu installation)
# Add Docker's official GPG key
```

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg \
  --dearmor -o /etc/apt/keyrings/docker.gpg

# Add Docker repository
echo "deb [arch=$(dpkg --print-architecture) \
  signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Install Docker
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io \
  docker-buildx-plugin docker-compose-plugin

# Add user to docker group (avoid sudo for docker commands)
sudo usermod -aG docker $USER

# 4. Install GitHub CLI
curl -fsSL https://cli.github.com/packages/githubcli-archive-keyring.gpg \
  | sudo dd of=/usr/share/keyrings/githubcli-archive-keyring.gpg
sudo chmod go+r /usr/share/keyrings/githubcli-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) \
  signed-by=/usr/share/keyrings/githubcli-archive-keyring.gpg] \
  https://cli.github.com/packages stable main" | \
  sudo tee /etc/apt/sources.list.d/github-cli.list > /dev/null
sudo apt update
sudo apt install -y gh

# 5. Logout and login again (or restart) to activate docker group membership
echo "□ Please logout and login again (or restart) to activate"
echo "   Docker permissions"
echo "After reboot, continue with the next section..."
```

After Reboot: Authentication Setup

```
# 1. Verify Docker works without sudo
docker run hello-world

# 2. Authenticate with GitHub CLI
```

```
gh auth login
# Follow prompts:
# - What account do you want to log into? GitHub.com
# - What is your preferred protocol? HTTPS
# - Authenticate Git with your GitHub credentials? Yes
# - How would you like to authenticate? Login with a web browser
# (Copy the one-time code, open browser, paste code, complete auth)

# 3. Verify GitHub authentication
gh auth status
```

Install ZZCOLLAB System

```
# 1. Clone and install zzcollab
git clone https://github.com/rgt47/zzcollab.git
cd zzcollab && ./install.sh

# 2. Add zzcollab to PATH (add to ~/.bashrc or ~/.zshrc)
echo 'export PATH="$HOME/bin:$PATH"' >> ~/.bashrc
source ~/.bashrc

# 3. Verify zzcollab installation
zzcollab --help && which zzcollab

# 4. Optional: Set up configuration for easier commands
zzcollab --config init
zzcollab --config set team-name "rgt47"          # Match team settings
zzcollab --config set dotfiles-dir "~/dotfiles"  # Your dotfiles path
```

Join Team Project (Standard Workflow)

```
# 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 team project
git clone https://github.com/rgt47/png1.git
cd png1

# 3. Set up development environment with team base image
```

```
# Try available interfaces (helpful errors if variant unavailable):

# With Configuration (Recommended):
zzcollab -p png1 -I shell      # Shell interface (command line development)
zzcollab -p png1 -I rstudio    # RStudio interface (web-based IDE at
                               # localhost:8787)
zzcollab -p png1 -I verse      # Publishing interface (LaTeX support for reports)

# Traditional Verbose Approach:
zzcollab -t rgt47 -p png1 -I shell -d ~/dotfiles      # Shell interface
zzcollab -t rgt47 -p png1 -I rstudio -d ~/dotfiles    # RStudio interface
zzcollab -t rgt47 -p png1 -I verse -d ~/dotfiles      # Publishing interface

# 4. Start development environment
make docker-zsh        # For shell interface
# OR
make docker-rstudio    # For RStudio interface (then visit localhost:8787)
# OR
make docker-verse      # For publishing interface

# 5. Verify everything works
R
# Test that you can load the project
devtools::load_all()   # Load all project functions
devtools::test()       # Run project tests
quit()
```

Development Workflow (Same as Other Platforms)

```
# Daily development cycle
make docker-zsh                    # Start development container
# ... do analysis work inside container ...
exit                              # Exit container

# Git workflow
git add .
git commit -m "Add my analysis with tests"
git push origin main               # Triggers automatic team env
                                   # updates
```

Troubleshooting Ubuntu-Specific Issues

```
# If Docker permission denied errors persist:
sudo systemctl restart docker
sudo usermod -aG docker $USER
# Then logout/login again

# If GitHub CLI authentication fails:
gh auth refresh --hostname github.com --scopes repo,read:org

# If zzcollab command not found:
echo $PATH  # Verify ~/bin is in PATH
ls ~/bin/zzcollab  # Verify zzcollab binary exists
chmod +x ~/bin/zzcollab  # Make executable if needed

# If Docker daemon not running:
sudo systemctl start docker
sudo systemctl enable docker  # Start automatically on boot
```

What This Ubuntu Setup Provides:

- [CHECKMARK] **Complete development environment**: Docker + GitHub + ZZCOLLAB
- [CHECKMARK] **Team integration ready**: Can immediately join existing projects

- [CHECKMARK] **Professional toolchain**: Same tools as macOS/Windows team members
- [CHECKMARK] **Zero configuration differences**: Identical development experience across platforms
- [CHECKMARK] **Enterprise security**: Proper user permissions and authentication

Ubuntu-Specific Advantages:

- [CHECKMARK] **Native Docker performance**: Better than Docker Desktop on macOS/Windows
- [CHECKMARK] **Package manager integration**: Official repositories for all tools
- [CHECKMARK] **Lightweight system**: More resources available for analysis containers

- [CHECKMARK] **Perfect for development**: Many data scientists prefer Linux environments

Once complete, **Developer 2** on Ubuntu has identical capabilities to team members on macOS or Windows! [ROCKET]

R Interface Alternative (Advanced)

For teams comfortable with R, ZZCOLLAB provides a complete R interface with configuration support:

```
# Method 1: Using Configuration (Recommended)
library(zzcollab)

# One-time setup for team lead
init_config()                                    # Initialize config file
set_config("team_name", "rgt47")                 # Set team name
set_config("build_mode", "standard")             # Set preferred mode
set_config("dotfiles_dir", "~/dotfiles")         # Set dotfiles path

# Team Lead (Developer 1) - Simplified with config
init_project(project_name = "png1")              # Uses config defaults

# Team Members (Dev 2 & 3) - Simplified with config
set_config("team_name", "rgt47")                 # Match team settings
join_project(project_name = "png1", interface = "shell")  # Uses config defaults

# Method 2: Traditional Explicit Parameters
library(zzcollab)

# Team Lead (Developer 1) - R Interface with explicit parameters
init_project(
  team_name = "rgt47",
  project_name = "png1",
  build_mode = "standard",
  dotfiles_path = "~/dotfiles"
)

# Team Members (Dev 2 & 3) - R Interface with explicit parameters
join_project(
```

```
  team_name = "rgt47",
  project_name = "png1",
  interface = "shell",
  build_mode = "fast",
  dotfiles_path = "~/dotfiles"
)
```

The R interface provides identical functionality to the command-line interface but within the familiar R environment. All configuration system benefits apply to the R interface as well.