

what is the set of commands for dev 1 if he wants to initiate a new analysis project for dev 2 and dev 3 to work on, , png1, without building a local workspace.

□ Based on the current zzcollab selective base image building system, here are the commands for Developer 1 (Team Lead) to initiate a new analysis project png1 for team collaboration without building a local workspace:

Developer 1: Team Lead Setup (No Local Workspace)

1. Navigate to projects directory `cd ~/projects` # or your preferred projects directory

2. Initialize team project 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`

Note: Omitting `-d ~/dotfiles` means no local workspace is built

What This Does:

1. Creates project directory: `png1/`
2. Sets up team Docker images: - Builds and pushes to Docker Hub as `rgt47/png1core-shell:latest` (and/or other variants)
3. Initializes zzcollab project structure: Complete R package with analysis framework
4. Creates private GitHub repository: `https://github.com/rgt47/png1`
5. Sets up automated CI/CD: GitHub Actions for team image management
6. Enables team collaboration: Dev 2 and Dev 3 can join immediately

For Dev 2 and Dev 3 to Join:

1. Clone the project `git clone https://github.com/rgt47/png1.git` `cd png1`

2. 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

3. Start development `make docker-zsh` # or `make docker-rstudio`,
`make docker-verse`

Key Benefits of This Approach:

- ☐ No local workspace for Dev 1: Team infrastructure created without personal development setup
- ☐ Faster initialization: Only builds needed variants, not all three
- ☐ Immediate team access: Dev 2 & 3 can join as soon as GitHub repo is created
- ☐ Flexible team scaling: Can add more variants later with `zzcollab -V rstudio`
- ☐ 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! ☐

what does dev 2 do when finished development?

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

Dev 2: Development Completion 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 - recom-
mended) 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 de-
scription]"
-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: - ☐ Runs R package validation - ☐ Executes all tests - ☐ Renders analysis reports - ☐ Detects if new packages were added
2. If new packages detected: - ☐ Rebuilds team Docker image with new packages - ☐ Pushes updated image to Docker Hub (rgt47/png1core-*.latest) - ☐ Notifies team via commit comment with update instructions
3. Team gets notification: ☐ 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 environ-  
ment 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:

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

Dev 2's Work is Done! ☐

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! ☐