

workflow_solo

August 06, 2025

ZZCOLLAB Solo Developer Workflow Guide

Initial Setup (One-Time)

1. Install ZZCOLLAB System

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

```
# Verify installation
zzcollab --help && which zzcollab
```

2. Configuration Setup (Recommended)

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

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

```
# Set your defaults (customize as needed)
```

```
zzcollab --config set team-name "rgt47"
```

```
zzcollab --config set github-account "rgt47"
```

```
zzcollab --config set build-mode "standard"
```

```
zzcollab --config set dotfiles-dir "~/dotfiles"
```

```
# Your Docker Hub account
```

```
# Your GitHub username
```

```
# fast, standard, comprehensive
```

```
# Path to your dotfiles
```

```
# View your configuration
```

```
zzcollab --config list
```

Project Creation

Choose Your Development Environment

Quick Start (Recommended):

```
# Creates optimal variants automatically + GitHub repo
zzcollab -i -p myproject --github # Creates: minimal + analysis variants
```

Interactive Variant Selection (Power Users):

```
mkdir myproject && cd myproject
zzcollab -i -p myproject # Creates project + config.yaml
./add_variant.sh # Browse comprehensive variant library
```

```
# Interactive menu shows 14 variants:
# □ STANDARD RESEARCH ENVIRONMENTS
# 1) minimal ~800MB - Essential R packages only
# 2) analysis ~1.2GB - Tidyverse + data analysis tools
# 3) modeling ~1.5GB - Machine learning with tidymodels
# 4) publishing ~3GB - LaTeX, Quarto, bookdown, blogdown
# 5) shiny ~1.8GB - Interactive web applications
# 6) shiny_verse ~3.5GB - Shiny with tidyverse + publishing
#
# □ SPECIALIZED DOMAINS
# 7) bioinformatics ~2GB - Bioconductor genomics packages
# 8) geospatial ~2.5GB - sf, terra, leaflet mapping tools
#
# □ LIGHTWEIGHT ALPINE VARIANTS
# 9) alpine_minimal ~200MB - Ultra-lightweight for CI/CD
# 10) alpine_analysis ~400MB - Essential analysis in tiny container
# 11) hpc_alpine ~600MB - High-performance parallel processing
#
# □ R-HUB TESTING ENVIRONMENTS
# 12) rhub_ubuntu ~1GB - CRAN-compatible package testing
# 13) rhub_fedora ~1.2GB - Test against R-devel
# 14) rhub_windows ~1.5GB - Windows compatibility testing
```

```
# Select variants that match your workflow, then:
zzcollab --variants-config config.yaml --github
```

Legacy Approach (Limited to 3 variants):

```
# Traditional: Limited to shell/rstudio/verse only
```

```
zzcollab -i -p myproject -B rstudio --github # RStudio interface
zzcollab -i -p myproject -B all --github      # All 3 legacy variants
```

Recommended Variant Combinations by Use Case:

- **Data Analysts:** analysis + publishing (tidyverse + reporting)
- **Bioinformaticians:** bioinformatics + alpine_minimal (research + CI/CD)
- **Package Developers:** minimal + rhub_ubuntu (development + testing)
- **Web App Developers:** shiny_verse + alpine_minimal (apps + deployment)
- **Academic Researchers:** modeling + publishing (analysis + manuscripts)

Daily Development Workflow

1. Start Development Environment

```
cd myproject
```

```
# Choose your interface based on selected variants:
```

```
make docker-zsh          # Shell interface (works with any variant)
make docker-rstudio      # RStudio Server at localhost:8787 (if rstudio/shiny variants)
make docker-r            # R console only
make docker-verse        # Publishing workflow with LaTeX (if publishing variant)
```

2. Iterative Development (Inside Container)

Working in the container:

```
# You're now inside the Docker container with all packages pre-installed
# Your project directory is mounted at /home/analyst/project
```

```
# R package development
```

```
R
devtools::load_all()      # Load your package functions
devtools::test()          # Run tests
devtools::document()     # Generate documentation
quit()
```

```
# Analysis scripts
```

```
vim scripts/01_data_analysis.R # Create/edit analysis
R --vanilla < scripts/01_data_analysis.R # Run script
```

```
# Create new functions
```

```

vim R/my_functions.R           # Add R functions
vim tests/testthat/test-my_functions.R # Write tests

# Install additional packages (will be tracked automatically)
R
install.packages("newpackage")
quit()

# Work on reports
vim analysis/report.Rmd       # Edit R Markdown report
R
rmarkdown::render("analysis/report.Rmd") # Generate report
quit()

# Git workflow (from inside container)
git status                    # Check changes
git add .                    # Stage changes
git diff --cached             # Review staged changes

```

3. Exit Container and Commit

```

# Exit the development container
exit

# You're now back on your host system
# Validate dependencies and run final tests
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

# Git workflow - commit and push
git status                    # Check what changed
git diff                      # Review changes

git add .
git commit -m "Add data analysis with visualization"

- Implement customer segmentation analysis
- Add clustering functions with tests
- Generate summary report with plots

```

- All tests passing, dependencies validated"

```
git push origin main          # Push to GitHub
```

4. What Happens Automatically

When you push changes to GitHub:

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 Docker image with new packages
- [CHECKMARK] Pushes updated image to Docker Hub
- [CHECKMARK] Next time you run `make docker-zsh`, you get the updated environment

5. Continue Development Cycle

```
# Start next iteration
make docker-zsh          # Continue with updated environment
# ... more development inside container ...
exit
# ... commit and push changes ...
```

Advanced Development Patterns

Working with Different Variants

```
# Switch between different environments for different tasks
make docker-zsh          # Use analysis variant for data exploration
make docker-rstudio      # Use RStudio for interactive development
make docker-verse        # Use publishing variant for report writing

# Each environment has specialized packages for its purpose
```

Testing and Validation Workflow

```
# Before committing, validate your work:
make docker-test         # Run all automated tests
```

```

make docker-check          # R CMD check validation
make docker-render         # Ensure all reports render

# Fix any issues before committing
make docker-zsh
# ... fix issues inside container ...
exit

```

Package Development Focus

```

# Inside container - R package development workflow
R
devtools::check()          # Full package check
devtools::build()          # Build package
devtools::install()        # Install your package
usethis::use_test("myfunction") # Create test file
quit()

# Document and check
make docker-document       # Generate documentation
make docker-check          # Full package validation

```

Benefits of This Workflow

- [CHECKMARK] **Reproducible:** Every development session uses identical environment
- [CHECKMARK] **Isolated:** No conflicts with your host system R installation
- [CHECKMARK] **Collaborative-ready:** Easy to share exact environment with others
- [CHECKMARK] **Professional:** Automated testing, validation, and CI/CD
- [CHECKMARK] **Flexible:** 14+ variants for different research domains
- [CHECKMARK] **Lightweight options:** Alpine variants ~200MB vs standard ~1GB+
- [CHECKMARK] **Automatic dependency tracking:** Never lose track of required packages
- [CHECKMARK] **Version controlled:** Complete project history including environment

Solo to Team Transition

If you later want to collaborate:

1. **Your project is already team-ready** - others can join with:

```
git clone https://github.com/yourname/myproject.git
cd myproject
zzcollab -t yourname -p myproject -I analysis # Join with analysis variant
```

2. **No migration needed** - the same Docker images and workflow work for teams

This workflow provides **enterprise-grade reproducibility** with **solo developer simplicity**! [ROCKET]

Practical Example: Penguin Analysis Project

Let's walk through a complete example using the Palmer penguins dataset to demonstrate the iterative development workflow.

Step 1: Create Project and Initial Analysis

```
# Set up the project
zzcollab -i -p penguin-analysis --github
```

```
# Start development environment
cd penguin-analysis
make docker-zsh
```

Inside the container - Create initial analysis script:

```
# Create the analysis script
vim scripts/01_penguin_exploration.R
```

Contents of scripts/01_penguin_exploration.R:

```
#' Penguin Bill Analysis
#' Explore relationship between bill depth and log of bill length

# Load required packages
library(palmerpenguins)
library(ggplot2)
library(dplyr)

#' Create scatter plot of bill depth vs log(bill length)
#' @return ggplot object
create_bill_plot <- function() {
```

```

penguins %>%
  filter(!is.na(bill_length_mm), !is.na(bill_depth_mm)) %>%
  ggplot(aes(x = log(bill_length_mm), y = bill_depth_mm)) +
  geom_point(aes(color = species), alpha = 0.7, size = 2) +
  labs(
    title = "Penguin Bill Depth vs Log(Bill Length)",
    x = "Log(Bill Length) (mm)",
    y = "Bill Depth (mm)",
    color = "Species"
  ) +
  theme_minimal() +
  theme(legend.position = "bottom")
}

# Create and display the plot
bill_plot <- create_bill_plot()
print(bill_plot)

# Save the plot
ggsave("figures/bill_analysis.png", bill_plot, width = 8, height = 6, dpi = 300)

cat("Analysis complete! Plot saved to figures/bill_analysis.png\n")

```

Create the function file:

```

# Create R function file
vim R/penguin_functions.R

```

Contents of R/penguin_functions.R:

```

#' Create scatter plot of bill depth vs log(bill length)
#'
#' @param data Data frame containing penguin data (default: palmerpenguins::penguins)
#' @return ggplot object
#' @export
#' @examples
#' plot <- create_bill_plot()
#' print(plot)
create_bill_plot <- function(data = palmerpenguins::penguins) {
  if (!requireNamespace("ggplot2", quietly = TRUE)) {
    stop("ggplot2 package is required")
  }
}

```



```

if (!requireNamespace("dplyr", quietly = TRUE)) {
  stop("dplyr package is required")
}

data %>%
  dplyr::filter(!is.na(bill_length_mm), !is.na(bill_depth_mm)) %>%
  ggplot2::ggplot(ggplot2::aes(x = log(bill_length_mm), y = bill_depth_mm)) +
  ggplot2::geom_point(ggplot2::aes(color = species), alpha = 0.7, size = 2) +
  ggplot2::labs(
    title = "Penguin Bill Depth vs Log(Bill Length)",
    x = "Log(Bill Length) (mm)",
    y = "Bill Depth (mm)",
    color = "Species"
  ) +
  ggplot2::theme_minimal() +
  ggplot2::theme(legend.position = "bottom")
}

```

Create tests for the function:

Create test file

vim tests/testthat/test-penguin_functions.R

Contents of tests/testthat/test-penguin_functions.R:

```

test_that("create_bill_plot works correctly", {
  # Test with default data
  plot <- create_bill_plot()

  # Check that it returns a ggplot object
  expect_s3_class(plot, "ggplot")

  # Check plot components
  expect_equal(plot$labels$title, "Penguin Bill Depth vs Log(Bill Length)")
  expect_equal(plot$labels$x, "Log(Bill Length) (mm)")
  expect_equal(plot$labels$y, "Bill Depth (mm)")
  expect_equal(plot$labels$colour, "Species")
})

test_that("create_bill_plot handles custom data", {
  # Create test data
  test_data <- data.frame(

```

```

    bill_length_mm = c(40, 45, 50),
    bill_depth_mm = c(18, 20, 22),
    species = c("A", "B", "C")
  )

  plot <- create_bill_plot(test_data)
  expect_s3_class(plot, "ggplot")
})

test_that("create_bill_plot handles missing values", {
  # Create test data with NA values
  test_data <- data.frame(
    bill_length_mm = c(40, NA, 50),
    bill_depth_mm = c(18, 20, NA),
    species = c("A", "B", "C")
  )

  plot <- create_bill_plot(test_data)
  expect_s3_class(plot, "ggplot")

  # Should have only 1 point after filtering NAs
  expect_equal(nrow(plot$data), 1)
})

```

Test and run the analysis:

```

# Install required packages
R
install.packages(c("palmerpenguins", "ggplot2", "dplyr"))
quit()

# Test the function
R
devtools::load_all()
devtools::test()
quit()

# Run the analysis script
mkdir -p figures
R --vanilla < scripts/01_penguin_exploration.R

```

```
# Check the git status
```

```
git status
```

```
git add .
```

```
git diff --cached
```

Step 2: Exit Container and First Commit

```
# Exit the development container
```

```
exit
```

```
# Validate dependencies and test
```

```
make docker-check-renv-fix # Auto-add new packages to renv.lock
```

```
make docker-test # Run tests in clean environment
```

```
# First commit and push
```

```
git add .
```

```
git commit -m "Add initial penguin bill analysis"
```

- Create scatter plot of bill depth vs log(bill length)
- Add create_bill_plot() function with comprehensive tests
- Generate publication-quality figure
- All tests passing, dependencies tracked"

```
git push origin main
```

Step 3: Continue Analysis - Add Regression Line

After the first push, continue with enhanced analysis:

```
# Start development environment again
```

```
make docker-zsh
```

```
# Update the analysis script
```

```
vim scripts/01_penguin_exploration.R
```

Updated scripts/01_penguin_exploration.R:

```
#' Penguin Bill Analysis - Enhanced with Regression
```

```
#' Explore relationship between bill depth and log of bill length
```

```
# Load required packages
```

```
library(palmerpenguins)
```

```

library(ggplot2)
library(dplyr)
library(broom)

#' Create enhanced scatter plot with regression line
#' @return ggplot object
create_enhanced_bill_plot <- function() {
  penguins %>%
    filter(!is.na(bill_length_mm), !is.na(bill_depth_mm)) %>%
    ggplot(aes(x = log(bill_length_mm), y = bill_depth_mm)) +
    geom_point(aes(color = species), alpha = 0.7, size = 2) +
    geom_smooth(method = "lm", se = TRUE, color = "black", linetype = "dashed") +
    labs(
      title = "Penguin Bill Depth vs Log(Bill Length) with Regression Line",
      x = "Log(Bill Length) (mm)",
      y = "Bill Depth (mm)",
      color = "Species",
      caption = "Dashed line shows linear regression fit with 95% confidence interval"
    ) +
    theme_minimal() +
    theme(legend.position = "bottom")
}

#' Fit linear model for bill depth vs log(bill length)
#' @return list with model object and summary statistics
fit_bill_model <- function() {
  clean_data <- penguins %>%
    filter(!is.na(bill_length_mm), !is.na(bill_depth_mm)) %>%
    mutate(log_bill_length = log(bill_length_mm))

  model <- lm(bill_depth_mm ~ log_bill_length, data = clean_data)

  list(
    model = model,
    summary = summary(model),
    r_squared = summary(model)$r.squared,
    coefficients = tidy(model)
  )
}

```

```

# Create enhanced plot
enhanced_plot <- create_enhanced_bill_plot()
print(enhanced_plot)

# Fit regression model
model_results <- fit_bill_model()
cat("\nRegression Results:\n")
cat("R-squared:", round(model_results$r_squared, 3), "\n")
print(model_results$coefficients)

# Save outputs
ggsave("figures/bill_analysis_with_regression.png", enhanced_plot,
       width = 8, height = 6, dpi = 300)

# Save model results
saveRDS(model_results, "results/bill_model.rds")

cat("\nEnhanced analysis complete!\n")
cat("Plot: figures/bill_analysis_with_regression.png\n")
cat("Model: results/bill_model.rds\n")

```

Update the function file:

```
vim R/penguin_functions.R
```

Add to R/penguin_functions.R:

```

#' Create enhanced scatter plot with regression line
#'
#' @param data Data frame containing penguin data (default: palmerpenguins::penguins)
#' @return ggplot object
#' @export
create_enhanced_bill_plot <- function(data = palmerpenguins::penguins) {
  if (!requireNamespace("ggplot2", quietly = TRUE)) {
    stop("ggplot2 package is required")
  }
  if (!requireNamespace("dplyr", quietly = TRUE)) {
    stop("dplyr package is required")
  }

  data %>%
    dplyr::filter(!is.na(bill_length_mm), !is.na(bill_depth_mm)) %>%

```

```

ggplot2::ggplot(ggplot2::aes(x = log(bill_length_mm), y = bill_depth_mm)) +
ggplot2::geom_point(ggplot2::aes(color = species), alpha = 0.7, size = 2) +
ggplot2::geom_smooth(method = "lm", se = TRUE, color = "black", linetype = "dashed")
ggplot2::labs(
  title = "Penguin Bill Depth vs Log(Bill Length) with Regression Line",
  x = "Log(Bill Length) (mm)",
  y = "Bill Depth (mm)",
  color = "Species",
  caption = "Dashed line shows linear regression fit with 95% confidence interval"
) +
ggplot2::theme_minimal() +
ggplot2::theme(legend.position = "bottom")
}

#' Fit linear model for bill depth vs log(bill length)
#'
#' @param data Data frame containing penguin data (default: palmerpenguins::penguins)
#' @return list with model object and summary statistics
#' @export
fit_bill_model <- function(data = palmerpenguins::penguins) {
  if (!requireNamespace("dplyr", quietly = TRUE)) {
    stop("dplyr package is required")
  }
  if (!requireNamespace("broom", quietly = TRUE)) {
    stop("broom package is required")
  }

  clean_data <- data %>%
    dplyr::filter(!is.na(bill_length_mm), !is.na(bill_depth_mm)) %>%
    dplyr::mutate(log_bill_length = log(bill_length_mm))

  model <- lm(bill_depth_mm ~ log_bill_length, data = clean_data)

  list(
    model = model,
    summary = summary(model),
    r_squared = summary(model)$r.squared,
    coefficients = broom::tidy(model)
  )
}

```

Add tests for new functions:

```
vim tests/testthat/test-penguin_functions.R
```

Add to test file:

```
test_that("create_enhanced_bill_plot works correctly", {
  plot <- create_enhanced_bill_plot()

  expect_s3_class(plot, "ggplot")
  expect_equal(plot$labels$title,
               "Penguin Bill Depth vs Log(Bill Length) with Regression Line")
  expect_true(grepl("regression", plot$labels$caption, ignore.case = TRUE))
})

test_that("fit_bill_model returns correct structure", {
  model_results <- fit_bill_model()

  expect_type(model_results, "list")
  expect_true("model" %in% names(model_results))
  expect_true("summary" %in% names(model_results))
  expect_true("r_squared" %in% names(model_results))
  expect_true("coefficients" %in% names(model_results))

  expect_s3_class(model_results$model, "lm")
  expect_type(model_results$r_squared, "double")
  expect_s3_class(model_results$coefficients, "data.frame")
})

test_that("fit_bill_model handles custom data", {
  test_data <- data.frame(
    bill_length_mm = c(40, 45, 50, 55),
    bill_depth_mm = c(18, 19, 20, 21),
    species = c("A", "B", "C", "A")
  )

  model_results <- fit_bill_model(test_data)
  expect_s3_class(model_results$model, "lm")
  expect_true(model_results$r_squared >= 0 && model_results$r_squared <= 1)
})
```

Test and run the enhanced analysis:

```

# Install new package
R
install.packages("broom")
quit()

# Test the new functions
R
devtools::load_all()
devtools::test()
quit()

# Create results directory and run enhanced analysis
mkdir -p results
R --vanilla < scripts/01_penguin_exploration.R

# Check what changed
git status
git diff

```

Step 4: Second Commit with Enhancement

```

# Exit container
exit

# Validate enhanced analysis
make docker-check-renv-fix    # Track new broom package
make docker-test              # Ensure all tests pass

# Commit the enhancement
git add .
git commit -m "Add regression analysis to penguin bill study"

- Add linear regression line to scatter plot
- Implement fit_bill_model() function with model diagnostics
- Include R-squared and coefficient estimates
- Add comprehensive tests for regression functionality
- Save model results for reproducibility
- All tests passing, broom package added to dependencies"

git push origin main

```


What This Example Demonstrates:

1. **Complete workflow:** From initial analysis to enhanced version
2. **Professional practices:** Functions, tests, documentation
3. **Iterative development:** Build on previous work incrementally
4. **Dependency tracking:** Automatic renv.lock updates
5. **Reproducible outputs:** Saved plots and model objects
6. **Quality assurance:** Tests validate function behavior
7. **Version control:** Clear commit messages with detailed changes

This example shows how ZZCOLLAB supports **professional data science workflows** with **minimal overhead!** ☐☐

Configuration System Benefits

With configuration set up, all commands become simpler:

Traditional verbose approach:

```
zzcollab -i -t rgt47 -p myproject -B analysis -S -d ~/dotfiles --github
```

Config-simplified approach (identical result):

```
zzcollab -i -p myproject -B analysis --github
```

Modern variant approach (uses config.yaml):

```
zzcollab -i -p myproject --github    # Creates default variants automatically
```

The configuration system eliminates repetitive typing while maintaining full flexibility for custom workflows.