MULTI_FILE_WORKFLOW

August 13, 2025 at 04:41 PM

Multi-File R Analysis Workflow with zzvim-R

This document demonstrates how to efficiently manage multiple R analyses across different files within a single Vim session using the zzvim-R plugin's buffer-specific terminal system.

Overview

The zzvim-R plugin provides **buffer-specific R terminals**, meaning each R file gets its own isolated R session. This enables:

- **Parallel Analysis**: Run different analyses simultaneously without variable conflicts
- **Context Preservation**: Each file maintains its own R workspace and loaded packages
- **Seamless Switching**: Move between analyses without losing state
- **Independent Sessions**: Errors in one analysis don't affect others

Core Concepts

Buffer-Specific Terminal Association

Each R file automatically gets its own terminal named `R-filename`:

- `analysis.R` → Terminal: `R-analysis`
- `data_prep.R` → Terminal: `R-data_prep`
- `visualization.R` → Terminal: `R-visualization`

Key Commands for Multi-File Workflows

Command	Purpose
`<LocalLeader>r`	Create/switch to buffer-specific R terminal
`<LocalLeader>w`	Open R terminal in vertical split
`<LocalLeader>W`	Open R terminal in horizontal split
`:RShowTerminal`	Show which terminal is associated with current buffer
`:RListTerminals`	List all file ↔ terminal associations
`:RSwitchToTerminal`	Jump to this buffer's R terminal

Example Workflow: Multi-Stage Data Analysis

Let's walk through a realistic scenario with three analysis files:

Step 1: Setup Project Structure

```
project/
├── 01_data_import.R     # Data loading and initial cleanup
├── 02_analysis.R        # Statistical analysis
└── 03_visualization.R   # Plots and reporting
```

Step 2: Start First Analysis - Data Import

1. **Open the first file:**

   ```
   vim 01_data_import.R
   ```

2. **File content example:**

   ```r
   # 01_data_import.R
   library(readr)
   library(dplyr)

   # Load raw data
   raw_data <- read_csv("data/sales_data.csv")

   # Initial data exploration
   str(raw_data)
   summary(raw_data)

   # Basic cleaning
   ```

```r
clean_data <- raw_data %>%
  filter(!is.na(sales_amount)) %>%
  mutate(date = as.Date(date))

# Save for next stage
saveRDS(clean_data, "data/clean_data.rds")
```

3. **Create R terminal and run analysis:**
   - Press `<LocalLeader>w` to open vertical split with R terminal
   - Press `<CR>` on each line/block to execute
   - The terminal is named `R-01_data_import`
4. **Verify terminal association:**

   ```
   :RShowTerminal
   ```

   Output: `Current buffer associated with terminal: R-01_data_import [running]`

Step 3: Start Second Analysis - Statistical Analysis

1. **Open second file in new buffer:**

   ```
   :edit 02_analysis.R
   ```

2. **File content example:**

   ```r
   # 02_analysis.R
   library(dplyr)
   library(ggplot2)
   library(broom)

   # Load cleaned data from previous step
   clean_data <- readRDS("data/clean_data.rds")

   # Exploratory analysis
   summary_stats <- clean_data %>%
     group_by(category) %>%
     summarise(
       mean_sales = mean(sales_amount),
       median_sales = median(sales_amount),
       count = n()
     )
   ```

```r
# Statistical modeling
model <- lm(sales_amount ~ category + date, data = clean_data)
model_summary <- tidy(model)

# Save results
saveRDS(model_summary, "results/model_results.rds")
```

3. **Create separate R terminal for this analysis:**
   - Press <LocalLeader>r to create new buffer-specific terminal
   - This creates R-02_analysis terminal (separate from the first)
   - Execute code with <CR> as before
4. **Check current terminal associations:**

   ```
   :RListTerminals
   ```

   Output:

   ```
   R File ↔ Terminal Associations:
   ================================
   01_data_import.R  → R-01_data_import [running]
   02_analysis.R     → R-02_analysis   [running]
   ```

Step 4: Start Third Analysis - Visualization

1. **Open third file:**

   ```
   :edit 03_visualization.R
   ```

2. **File content example:**

   ```r
   # 03_visualization.R
   library(ggplot2)
   library(plotly)
   library(dplyr)

   # Load data and model results
   clean_data <- readRDS("data/clean_data.rds")
   model_results <- readRDS("results/model_results.rds")

   # Create visualizations
   sales_plot <- ggplot(clean_data, aes(x = date, y = sales_amount, color = category)
     geom_point() +
     geom_smooth(method = "lm") +
   ```

```
  labs(title = "Sales Trends by Category")

# Interactive plot
interactive_plot <- ggplotly(sales_plot)

# Save plots
ggsave("plots/sales_trends.png", sales_plot)
```

3. **Create third R terminal:**
   - Press `<LocalLeader>w` for split terminal view
   - Creates `R-03_visualization` terminal
4. **View all active sessions:**

   ```
   :RListTerminals
   ```

   Output:

   ```
   R File ↔ Terminal Associations:
   ================================
   01_data_import.R   → R-01_data_import   [running]
   02_analysis.R      → R-02_analysis      [running]
   03_visualization.R → R-03_visualization [running]
   ```

Switching Between Analyses

Method 1: Using Buffer Commands

```
:buffer 01_data_import.R    " Switch to data import file
:buffer 02_analysis.R       " Switch to analysis file
:buffer 03_visualization.R  " Switch to visualization file
```

When you switch buffers, the plugin automatically associates you with the correct R terminal.

Method 2: Using Window Splits

1. **Split window to see multiple files:**

   ```
   :split 01_data_import.R     " Horizontal split
   :vsplit 02_analysis.R       " Vertical split
   ```

2. **Navigate between windows:**
   - `Ctrl-w h/j/k/l` to move between windows
   - Each window maintains its buffer-specific terminal association

Method 3: Using Terminal Window Jumping

1. **From any R file, jump to its terminal:**

   ```
   :RSwitchToTerminal
   ```

2. **See what's running in each terminal:**

   ```
   :RListTerminals
   ```

Advanced Multi-File Techniques

Sharing Data Between Sessions

Since each R terminal is independent, use file-based data sharing:

In 01_data_import.R:

```r
# Save processed data
saveRDS(clean_data, "shared/clean_data.rds")
write_csv(summary_table, "shared/summary.csv")
```

In 02_analysis.R:

```r
# Load shared data
clean_data <- readRDS("shared/clean_data.rds")
summary_table <- read_csv("shared/summary.csv")
```

Terminal Management Commands

Check status of specific terminal:

```
:RShowTerminal          " Show current buffer's terminal
```

List all R sessions:

```
:RListTerminals         " See all file ↔ terminal associations
```

Create new terminal split:

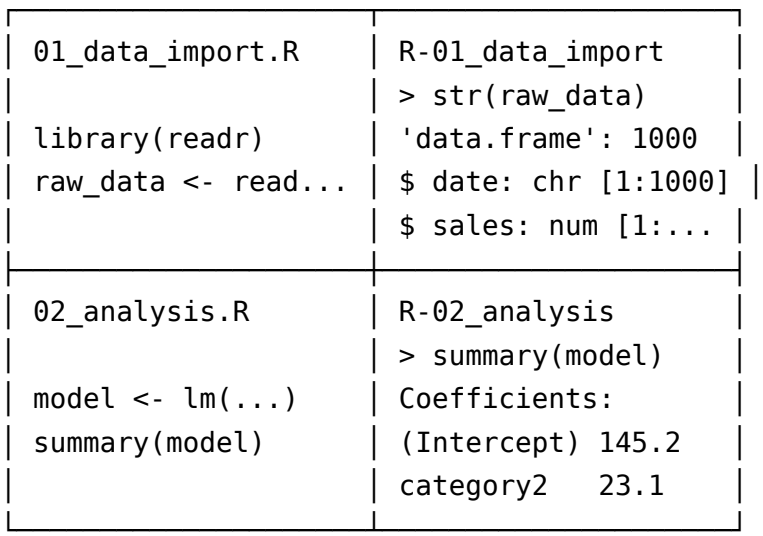
```
:ROpenSplit vertical    " Vertical split
:ROpenSplit horizontal  " Horizontal split
```

Workflow Optimization Tips

1. **Use descriptive filenames** - Terminal names are based on filename
2. **Keep related code together** - Each file should be a logical unit
3. **Save intermediate results** - Use RDS/CSV for data passing
4. **Monitor memory usage** - Each R session uses memory independently

5. **Use split windows** - View multiple files and terminals simultaneously

Example Session Layout



Common Patterns

Pattern 1: Sequential Analysis Pipeline

```
" Open files in sequence
:edit 01_data_import.R    " Create R-01_data_import
:edit 02_analysis.R       " Create R-02_analysis
:edit 03_visualization.R  " Create R-03_visualization
```

Pattern 2: Parallel Development

```
" Work on multiple analyses simultaneously
:vsplit analysis_a.R      " Create R-analysis_a
:split analysis_b.R       " Create R-analysis_b
```

Pattern 3: Interactive Exploration

```
" Quick exploration in dedicated file
:edit scratch.R           " Create R-scratch for experiments
```

Troubleshooting

Problem: Terminal Not Responding

```
:RShowTerminal           " Check terminal status
:RListTerminals          " View all terminals
```

Problem: Wrong Terminal Association

```
" Force new terminal creation
<LocalLeader>r           " Creates new buffer-specific terminal
```

Problem: Too Many Open Terminals

```
:RListTerminals          " See all associations
" Close unnecessary terminals manually in terminal windows
```

Key Benefits of This Workflow

1. **Isolation**: Each analysis has its own workspace and package environment
2. **Parallelism**: Run long computations in one file while working on another
3. **Organization**: Clear separation of concerns across files
4. **State Preservation**: Switch between files without losing R session state
5. **Flexibility**: Mix interactive exploration with structured analysis
6. **Error Resilience**: Problems in one analysis don't crash others

This multi-file workflow transforms Vim into a powerful R development environment that rivals dedicated IDEs while maintaining the speed and efficiency of terminal-based editing.