Linux-Editors

K19G

2025-08-07

Table of contents

Preface **14**
Structure . . . 14
How to Use This Book . . . 14
Prerequisites . . . 14
Acknowledgments . . . 14

Intro **15**

Comprehensive Vi Editor Tutorial **17**
Table of Contents . . . 17
Introduction to Vi . . . 17
Starting Vi . . . 17
Modes in Vi . . . 18
Basic Navigation . . . 18
Editing Text . . . 18
Saving and Exiting . . . 19
Search and Replace . . . 19
Searching . . . 19
Replacing . . . 19
Copy, Cut, and Paste . . . 19
Undo and Redo . . . 19
Advanced Features . . . 20
Multiple Files . . . 20
Splitting Windows (Vim-specific) . . . 20
Macros . . . 20
Command Repetition . . . 20
Customizing Vi . . . 20
Tips and Tricks . . . 21
Common Issues and Solutions . . . 21
Conclusion . . . 21

Comprehensive Nano Editor Tutorial **22**
Table of Contents . . . 22
Introduction to Nano . . . 22
Starting Nano . . . 22
Basic Navigation . . . 23
Editing Text . . . 23
Saving and Exiting . . . 23
Search and Replace . . . 23
Searching . . . 23

Replacing . . . 24
Copy, Cut, and Paste . . . 24
Undo and Redo . . . 24
Advanced Features . . . 24
Multiple Files . . . 24
Syntax Highlighting . . . 24
Executing Commands . . . 25
Spell Checking . . . 25
Customizing Nano . . . 25
Tips and Tricks . . . 25
Common Issues and Solutions . . . 25
Conclusion . . . 26

Vi **27**

Vi: The Original Visual Editor **28**
Why Learn Vi? . . . 28
Vi Modes . . . 28
1. Command Mode (Normal Mode) . . . 28
2. Insert Mode . . . 28
3. Command-Line Mode . . . 28
Basic Operations . . . 29
Starting Vi . . . 29
Exiting Vi . . . 29
Navigation Commands . . . 30
Basic Movement . . . 30
Advanced Navigation . . . 30
Text Editing Commands . . . 31
Inserting Text . . . 31
Deleting Text . . . 31
Changing Text . . . 31
Copying and Pasting . . . 32
Working with Multiple Files . . . 32
Search and Replace . . . 32
Basic Search . . . 32
Search and Replace . . . 33
Advanced Features . . . 33
Marks and Bookmarks . . . 33
Registers . . . 33
Macros . . . 34
Configuration . . . 34
Vi Configuration File (.exrc) . . . 34
Environment Variables . . . 34
Real-World Scenarios . . . 35
Scenario 1: Emergency Linux System Recovery . . . 35
Scenario 2: Linux Log File Analysis . . . 35
Scenario 3: Linux System Configuration . . . 36

Vi vs Modern Editors . . . 37
Advantages of Vi . . . 37
Limitations . . . 37
Tips and Best Practices . . . 37
Essential Tips . . . 37
Common Mistakes . . . 38
Productivity Shortcuts . . . 38
Troubleshooting . . . 38
Common Issues . . . 38
Recovery Options . . . 39

Vim **40**

Vim: Vi Improved **41**
Why Choose Vim? . . . 41
Installation . . . 41
Linux Distributions . . . 41
Building from Source . . . 42
Vim Modes . . . 42
1. Normal Mode . . . 42
2. Insert Mode . . . 42
3. Visual Mode . . . 42
4. Command-Line Mode . . . 42
5. Replace Mode . . . 43
Enhanced Navigation . . . 43
Advanced Movement . . . 43
Window and Tab Management . . . 43
Advanced Text Editing . . . 44
Visual Mode Operations . . . 44
Advanced Copy/Paste . . . 45
Text Objects . . . 45
Configuration and Customization . . . 46
Vimrc Configuration . . . 46
Advanced Vimrc Features . . . 48
Plugin Management . . . 49
Using vim-plug . . . 49
Installing Plugins . . . 50
Programming with Vim . . . 51
Code Navigation . . . 51
Code Completion . . . 51
Debugging Integration . . . 51
Real-World Development Scenarios . . . 52
Scenario 1: Python Development Setup . . . 52
Scenario 2: Web Development Workflow . . . 52
Scenario 3: Linux System Administration . . . 53
Advanced Vim Techniques . . . 53
Macros and Automation . . . 53

Regular Expressions . . . 54
Custom Commands . . . 54
Performance Optimization . . . 54
Vim Performance Tips . . . 54
Large File Handling . . . 55
Troubleshooting Common Issues . . . 55
Performance Problems . . . 55
Plugin Conflicts . . . 55
Recovery and Backup . . . 56

Neovim **57**

Neovim: The Future of Vim **58**
Why Choose Neovim? . . . 58
Installation . . . 58
Package Managers . . . 58
From Source . . . 59
AppImage (Portable) . . . 59
Configuration Structure . . . 59
Basic Lua Configuration . . . 60
init.lua Structure . . . 60
Plugin Management with Packer . . . 61
Installing Packer . . . 61
Packer Configuration . . . 61
Loading Plugins in init.lua . . . 64
Language Server Protocol (LSP) Setup . . . 64
LSP Configuration . . . 64
Installing Language Servers . . . 66
Tree-sitter Configuration . . . 66
Tree-sitter Setup . . . 66
Advanced Neovim Features . . . 67
Telescope Configuration . . . 67
Terminal Integration . . . 68
Real-World Development Setups . . . 69
Python Development . . . 69
JavaScript/TypeScript Development . . . 70
Go Development . . . 71
Debugging with DAP . . . 71
DAP Configuration . . . 71
Performance Optimization . . . 72
Startup Optimization . . . 72
Migration from Vim . . . 73
Compatibility Layer . . . 73
Troubleshooting . . . 74
Common Issues . . . 74

Nano **75**

Nano: The User-Friendly Terminal Editor **76**
Why Choose Nano? . . . 76
Installation . . . 76
Most Linux Distributions (Pre-installed) . . . 76
Additional Linux Distributions . . . 77
Basic Usage . . . 77
Starting Nano . . . 77
Interface Overview . . . 77
Essential Commands . . . 78
File Operations . . . 78
Navigation . . . 78
Text Editing . . . 79
Search and Replace . . . 79
Configuration . . . 80
Global Configuration . . . 80
Sample .nanorc Configuration . . . 80
Syntax Highlighting . . . 82
Custom Syntax Highlighting . . . 83
Advanced Features . . . 83
Multiple Buffers . . . 83
Spell Checking . . . 83
Backup Files . . . 84
Mouse Support . . . 84
Real-World Usage Scenarios . . . 84
Scenario 1: Quick Configuration File Edit . . . 84
Scenario 2: System Log Analysis . . . 85
Scenario 3: Script Development . . . 85
Scenario 4: Crontab Editing . . . 86
Command Line Options . . . 86
Useful Nano Options . . . 86
Environment Variables . . . 87
Tips and Best Practices . . . 87
Productivity Tips . . . 87
Common Shortcuts Summary . . . 87
Customization for Different Use Cases . . . 88
Troubleshooting . . . 89
Common Issues . . . 89
Recovery Options . . . 90

Emacs **91**

Emacs: The Extensible Editor **92**
Why Choose Emacs? . . . 92
Installation . . . 92
Linux Distributions . . . 92

Additional Linux Distributions . . . 93
Building from Source . . . 93
Basic Concepts . . . 93
Key Notation . . . 93
Buffers, Windows, and Frames . . . 93
Major and Minor Modes . . . 94
Essential Commands . . . 94
Starting and Exiting . . . 94
File Operations . . . 94
Buffer Management . . . 95
Window Management . . . 95
Navigation and Editing . . . 96
Basic Movement . . . 96
Advanced Navigation . . . 96
Text Editing . . . 97
Selection (Regions) . . . 97
Search and Replace . . . 98
Basic Search . . . 98
Replace Operations . . . 98
Advanced Search . . . 98
Configuration . . . 99
Init File . . . 99
Package Management . . . 101
Popular Packages . . . 101
Programming with Emacs . . . 102
Language-Specific Modes . . . 102
Code Navigation . . . 103
Debugging . . . 104
Org Mode . . . 104
Basic Org Syntax . . . 104
Org Mode Commands . . . 105
Org Configuration . . . 105
Real-World Development Scenarios . . . 106
Scenario 1: Python Development Setup . . . 106
Scenario 2: Web Development Workflow . . . 107
Scenario 3: System Administration . . . 107
Advanced Emacs Features . . . 108
Macros . . . 108
Registers . . . 108
Dired (Directory Editor) . . . 109
Shell Integration . . . 109
Customization and Themes . . . 109
Custom Themes . . . 109
Custom Functions . . . 110
Performance Optimization . . . 111
Startup Optimization . . . 111
Large File Handling . . . 111

Troubleshooting . . . 111
Common Issues . . . 111
Recovery . . . 112

Lazyvim **113**

LazyVim: Modern Neovim Configuration **114**
What is LazyVim? . . . 114
Why Choose LazyVim? . . . 114
Installation . . . 114
Prerequisites . . . 114
Required Dependencies . . . 115
Installation Steps . . . 115
Initial Setup and Configuration . . . 116
First Launch . . . 116
Basic Configuration Structure . . . 116
Basic Options Configuration . . . 116
Custom Keymaps . . . 118
Auto Commands . . . 119
Plugin Configuration . . . 121
Adding Custom Plugins . . . 121
Language-Specific Configurations . . . 122
Essential LazyVim Features . . . 124
File Explorer (Neo-tree) . . . 124
Fuzzy Finding (Telescope) . . . 124
LSP Features . . . 125
Git Integration (Lazygit) . . . 126
Terminal Integration . . . 126
Real-World Development Setups . . . 127
Python Development . . . 127
Web Development . . . 129
Go Development . . . 131
Customization Examples . . . 133
Custom Colorscheme . . . 133
Custom Dashboard . . . 134
Performance Optimization . . . 136
Startup Optimization . . . 136
Memory Optimization . . . 137
Troubleshooting . . . 139
Common Issues . . . 139
Reset Configuration . . . 139
Performance Issues . . . 140

Helix **141**

Helix: A Modern Modal Editor **142**
What is Helix? . . . 142
Why Choose Helix? . . . 142
Installation . . . 142
Package Managers . . . 142
Building from Source . . . 143
Post-Installation Setup . . . 143
Basic Concepts . . . 144
Selection-First Editing . . . 144
Multiple Selections . . . 144
Modes . . . 144
Essential Commands . . . 145
Starting Helix . . . 145
Basic Navigation . . . 145
Text Selection . . . 146
Editing Commands . . . 146
Search and Replace . . . 147
File and Buffer Management . . . 148
File Operations . . . 148
File Picker and Fuzzy Finding . . . 148
Configuration . . . 148
Configuration File . . . 148
Language Configuration . . . 151
Themes . . . 152
Custom Theme . . . 153
Real-World Usage Scenarios . . . 154
Scenario 1: Python Development . . . 154
Scenario 2: Web Development . . . 154
Scenario 3: Configuration File Editing . . . 155
Scenario 4: Log File Analysis . . . 155
Advanced Features . . . 156
Multiple Selections Mastery . . . 156
Text Objects . . . 156
Tree-sitter Navigation . . . 157
LSP Features . . . 157
Customization and Scripting . . . 158
Custom Commands . . . 158
Integration with External Tools . . . 158
Shell Integration . . . 158
Performance and Optimization . . . 159
Large File Handling . . . 159
Memory Usage . . . 159
Troubleshooting . . . 160
Common Issues . . . 160
Debug Mode . . . 160
Reset Configuration . . . 160

Migration from Other Editors . . . 160
From Vim/Neovim . . . 160
From VSCode . . . 161
From Kakoune . . . 161

Micro **162**

Micro: A Modern Terminal Editor **163**
What is Micro? . . . 163
Why Choose Micro? . . . 163
Installation . . . 163
Package Managers . . . 163
Direct Download . . . 164
From Source . . . 164
Basic Usage . . . 165
Starting Micro . . . 165
Interface Overview . . . 165
Essential Keybindings . . . 166
File Operations . . . 166
Navigation . . . 166
Text Editing . . . 166
Multiple Cursors . . . 167
Search and Replace . . . 167
Configuration . . . 167
Settings File . . . 167
Common Configuration Changes . . . 169
Keybinding Customization . . . 169
Color Schemes . . . 170
Custom Color Scheme . . . 170
Plugin System . . . 171
Plugin Management . . . 171
Popular Plugins . . . 171
Plugin Configuration . . . 172
Real-World Usage Scenarios . . . 172
Scenario 1: Linux System Configuration . . . 172
Scenario 2: Python Development . . . 173
Scenario 3: Log File Analysis . . . 174
Scenario 4: Web Development . . . 174
Advanced Features . . . 175
Command Mode . . . 175
Macros . . . 176
Bookmarks . . . 176
Splits and Tabs . . . 176
Integration with Development Tools . . . 177
Git Integration . . . 177
Language Server Integration . . . 177
Build System Integration . . . 177

Customization Examples . . . 178
Custom Status Line . . . 178
Custom Key Bindings for Productivity . . . 178
File Type Specific Settings . . . 178
Performance and Optimization . . . 179
Large File Handling . . . 179
Memory Usage . . . 179
Troubleshooting . . . 180
Common Issues . . . 180
Debug Mode . . . 180
Reset Configuration . . . 180
Migration from Other Editors . . . 180
From Nano . . . 180
From GUI Editors . . . 181
From Vim . . . 181

Kakoune **182**

Kakoune: The Selection-Based Editor **183**
What is Kakoune? . . . 183
Why Choose Kakoune? . . . 183
Installation . . . 183
Package Managers . . . 183
Building from Source . . . 184
Development Version . . . 184
Basic Concepts . . . 185
Selection-First Philosophy . . . 185
Multiple Selections . . . 185
Modes . . . 185
Essential Commands . . . 185
Starting Kakoune . . . 185
Basic Navigation . . . 186
Text Selection . . . 187
Multiple Selections . . . 187
Editing Commands . . . 188
Search and Replace . . . 188
File and Buffer Management . . . 189
Buffer Operations . . . 189
File Operations . . . 189
Window Management . . . 190
Configuration . . . 190
Configuration File . . . 190
Plugin Management . . . 191
Popular Plugins . . . 192
Real-World Usage Scenarios . . . 193
Scenario 1: Refactoring Variable Names . . . 193
Scenario 2: Multiple Line Editing . . . 194

Scenario 3: Complex Text Manipulation . . . 194
Scenario 4: Log File Analysis . . . 195
Advanced Features . . . 195
Hooks and Automation . . . 195
Custom Commands . . . 196
Text Objects . . . 196
Macros and Repetition . . . 197
Language-Specific Configurations . . . 197
Python Development . . . 197
Web Development . . . 198
Go Development . . . 199
Integration with External Tools . . . 199
Git Integration . . . 199
Build System Integration . . . 200
Terminal Integration . . . 201
Performance and Optimization . . . 201
Large File Handling . . . 201
Memory Optimization . . . 202
Troubleshooting . . . 202
Common Issues . . . 202
Session Management . . . 202
Configuration Issues . . . 203

Modern Tools **204**

Modern Linux Development Tools **205**
Why Modern Tools on Linux? . . . 205
Terminal-Based Development Tools . . . 205
Zellij: Modern Terminal Multiplexer . . . 205
Starship: Cross-Shell Prompt . . . 206
Zoxide: Smart Directory Navigation . . . 209
Eza: Modern ls Replacement . . . 210
Bat: Better Cat . . . 211
Ripgrep: Fast Text Search . . . 211
Fd: Better Find . . . 212
Delta: Better Git Diff . . . 213
File Management Tools . . . 214
Broot: Interactive Directory Navigation . . . 214
Nnn: Terminal File Manager . . . 215
Development Environment Tools . . . 216
Direnv: Environment Management . . . 216
Mise: Runtime Version Manager . . . 217
Just: Command Runner . . . 219
Text Processing Tools . . . 221
Jq: JSON Processor . . . 221
Yq: YAML Processor . . . 222
Sd: Sed Alternative . . . 223

Monitoring and System Tools 223
Htop: Process Viewer 223
Bottom: System Monitor 224
Procs: Process Information 225
Linux-Specific Integration 225
Systemd Integration 225
Linux Distribution Package Management 226
Linux Desktop Integration 227
Integration Examples 228
Complete Development Setup 228
Project Template with Modern Tools 229
Linux-Specific Troubleshooting 231
Permission Issues 231
Package Manager Conflicts 232
Performance Optimization for Linux 232
Container Integration 233

Preface

Welcome to this book!

Structure

This book is organized into categories and subcategories to help you navigate the content effectively.

How to Use This Book

You can read this book in various formats:

- Online HTML version
- Downloadable PDF
- EPUB for e-readers

Prerequisites

List any prerequisites or requirements here.

Acknowledgments

Add acknowledgments here.

Intro

Comprehensive Vi Editor Tutorial

Vi is a powerful, lightweight, and ubiquitous text editor in Linux and Unix systems. This tutorial covers the essentials of using Vi, from basic navigation to advanced editing techniques, suitable for beginners and intermediate users.

Table of Contents

1. Introduction to Vi
2. Starting Vi
3. Modes in Vi
4. Basic Navigation
5. Editing Text
6. Saving and Exiting
7. Search and Replace
8. Copy, Cut, and Paste
9. Undo and Redo
10. Advanced Features
11. Customizing Vi
12. Tips and Tricks
13. Common Issues and Solutions

Introduction to Vi

Vi (and its enhanced version, Vim, short for "Vi IMproved") is a highly configurable, modal text editor available on almost all Linux and Unix systems. Its efficiency comes from keyboard-driven commands, making it a favorite for developers and system administrators.

Key features: - Lightweight and fast - Modal editing (command, insert, visual modes) - Extensive plugin support (in Vim) - Available by default on most Linux distributions

This tutorial focuses on Vi commands, which are largely compatible with Vim.

Starting Vi

To start Vi, open a terminal and use the following commands:

- Open a file: `vi filename`
- Open a new file: `vi newfile.txt`

- Open in read-only mode: `view filename`
- Start Vim (if installed): `vim filename`

If the file exists, Vi opens it; otherwise, it creates a new file.

Modes in Vi

Vi operates in three main modes:

1. **Command Mode**: Default mode for navigation and issuing commands. Most keys trigger actions (e.g., `dd` deletes a line).
2. **Insert Mode**: For typing and editing text. Enter this mode with keys like `i` or `a`.
3. **Visual Mode**: For selecting text blocks. Enter with `v` (character-wise), `V` (line-wise), or `Ctrl+v` (block-wise).

Switch between modes: - **Command to Insert**: `i` (insert before cursor), `a` (append after cursor), `o` (new line below) - **Insert to Command**: `Esc` - **Command to Visual**: `v`, `V`, or `Ctrl+v` - **Visual to Command**: `Esc`

Basic Navigation

In Command Mode, use these keys to move the cursor:

- **Arrow keys**: Move up, down, left, right
- **h, j, k, l**: Left, down, up, right (vim-preferred)
- **w**: Move to the start of the next word
- **b**: Move to the start of the previous word
- **0**: Move to the start of the line
- **$**: Move to the end of the line
- **gg**: Go to the first line
- **G**: Go to the last line
- **:n**: Go to line number `n` (e.g., `:10` for line 10)

Pro Tip: Combine with numbers (e.g., `5j` moves down 5 lines).

Editing Text

Enter Insert Mode to edit text: - `i`: Insert before cursor - `a`: Append after cursor - `o`: Open a new line below - `O`: Open a new line above - `I`: Insert at the start of the line - `A`: Append at the end of the line

In Command Mode, edit with: - `x`: Delete character under cursor - `dw`: Delete word - `dd`: Delete current line - `D`: Delete from cursor to end of line - `r`: Replace single character - `R`: Replace multiple characters (overwrite mode) - `cw`: Change word (delete word and enter Insert Mode)

Use numbers for repetition (e.g., `3dd` deletes three lines).

Saving and Exiting

In Command Mode, use these commands (start with `:`): - `:w`: Save (write) the file - `:w filename`: Save as a new file - `:q`: Quit (if no changes made) - `:q!`: Quit without saving - `:wq` or `ZZ`: Save and quit - `:x`: Save and quit (only writes if changes were made)

Note: If you lack permissions, use `:w!` to force save (if possible).

Search and Replace

Searching

- `/pattern`: Search forward for `pattern`
- `?pattern`: Search backward for `pattern`
- `n`: Move to next match
- `N`: Move to previous match

Example: `/hello` searches for "hello". Use `n` to jump to the next occurrence.

Replacing

- `:%s/old/new/g`: Replace all occurrences of `old` with `new` in the file
- `:%s/old/new/gc`: Replace with confirmation for each occurrence
- `:s/old/new`: Replace in the current line only

Example: `:%s/foo/bar/g` replaces all "foo" with "bar".

Copy, Cut, and Paste

Vi uses "yank" for copying and "delete" for cutting: - `yy`: Yank (copy) current line - `yw`: Yank word - `p`: Paste after cursor - `P`: Paste before cursor - `dd`: Cut (delete) current line

Use numbers (e.g., `3yy` copies three lines). In Visual Mode: - Select text with `v`, `V`, or `Ctrl+v` - `y`: Copy selection - `d`: Cut selection - `p`: Paste after cursor

Undo and Redo

- `u`: Undo last change
- `Ctrl+r`: Redo undone change
- `:u`: Undo (alternative)

Note: Vi has a limited undo history; Vim supports more extensive undo.

Advanced Features

Multiple Files

- Open multiple files: `vi file1 file2`
- Switch files: `:n` (next), `:p` (previous)
- List open files: `:ls`
- Open another file: `:e filename`

Splitting Windows (Vim-specific)

- `:sp`: Split horizontally
- `:vsp`: Split vertically
- `Ctrl+w` followed by arrow keys: Switch between windows

Macros

- `qa`: Start recording macro to register `a`
- Perform actions
- `q`: Stop recording
- `@a`: Replay macro
- `@@`: Replay last macro

Command Repetition

- `.`: Repeat the last change
- Example: If you type `dw` to delete a word, `.` repeats it.

Customizing Vi

Vi and Vim can be customized via a configuration file: - For Vi: `~/.exrc` - For Vim: `~/.vimrc`

Example `.vimrc`:

```
set number          " Show line numbers
set tabstop=4       " Tabs are 4 spaces
set autoindent      " Enable auto-indentation
syntax on           " Enable syntax highlighting (Vim)
```

To apply settings in a session: - `:set number`: Enable line numbers - `:set nonumber`: Disable line numbers

Tips and Tricks

- **Jump to matching brace**: Use `%` on `{}`, `()`, or `[]`
- **Case-insensitive search**: `:set ignorecase`
- **Highlight search results**: `:set hlsearch`
- **Clear highlights**: `:noh`
- **Execute shell command**: `:!command` (e.g., `:!ls`)
- **Auto-complete (Vim)**: `Ctrl+n` or `Ctrl+p` in Insert Mode
- **Visual Block Mode**: `Ctrl+v` for column editing (e.g., insert text across multiple lines)

Common Issues and Solutions

1. **Stuck in Insert Mode**:
   - Press `Esc` to return to Command Mode.
2. **Permission Denied on Save**:
   - Use `:w!` to force save (if you have sudo privileges, try `:w !sudo tee %`).
3. **Accidental Macro Recording**:
   - Stop with `q` and avoid using `@` until you clear the register.
4. **Lost Changes**:
   - Check for swap files (e.g., `.filename.swp`). Recover with `vi -r filename`.
5. **Slow Performance**:
   - Disable syntax highlighting (`:syntax off`) or use Vi instead of Vim for large files.

Conclusion

Vi is a versatile editor that rewards practice. Start with basic navigation and editing, then explore advanced features like macros and splits as needed. For more functionality, consider using Vim, which extends Vi with additional features.

For further learning: - Run `vimtutor` in the terminal for an interactive Vim tutorial. - Check the Vi/Vim man page: `man vi` or `man vim`. - Explore online resources like the Vim documentation or community forums.

Comprehensive Nano Editor Tutorial

Nano is a user-friendly, lightweight text editor commonly found in Linux and Unix systems. This tutorial provides a comprehensive guide to using Nano, covering everything from basic operations to advanced features, suitable for beginners and intermediate users.

Table of Contents

1. Introduction to Nano
2. Starting Nano
3. Basic Navigation
4. Editing Text
5. Saving and Exiting
6. Search and Replace
7. Copy, Cut, and Paste
8. Undo and Redo
9. Advanced Features
10. Customizing Nano
11. Tips and Tricks
12. Common Issues and Solutions

Introduction to Nano

Nano is a simple, intuitive text editor designed for ease of use, especially for users new to Linux. Unlike Vi, Nano is non-modal, meaning you can start typing immediately without switching modes. It displays key commands at the bottom of the screen, making it beginner-friendly.

Key features: - Easy-to-use interface with on-screen shortcuts - Syntax highlighting (optional) - Search and replace functionality - Available on most Linux distributions

Starting Nano

To start Nano, open a terminal and use the following commands: - Open a file: `nano filename` - Open a new file: `nano newfile.txt` - Open with line numbers: `nano -l filename` - Open in read-only mode: `nano -v filename`

If the file exists, Nano opens it; otherwise, it creates a new file. The interface shows the file content with a status bar at the top and command shortcuts at the bottom.

Basic Navigation

Nano uses standard keyboard controls for navigation: - **Arrow keys**: Move up, down, left, right - **Ctrl+F**: Scroll forward one page - **Ctrl+B**: Scroll backward one page - **Ctrl+Y**: Scroll up one line - **Ctrl+V**: Scroll down one line - **Ctrl+_** (underscore): Go to a specific line number - **Home**: Move to the start of the line - **End**: Move to the end of the line - **Ctrl+A**: Move to the start of the line - **Ctrl+E**: Move to the end of the line

Note: `Ctrl` commands are shown as `^` in Nano's interface (e.g., `^G` for `Ctrl+G`).

Editing Text

Nano allows direct text editing without modes: - Type to insert text at the cursor position - **Backspace** or **Delete**: Remove characters - **Ctrl+K**: Cut the entire line - **Ctrl+U**: Paste the cut text - **Alt+6**: Copy the current line (not available in all versions) - **Ctrl+D**: Delete character under cursor - **Ctrl+H**: Delete character before cursor

Use arrow keys to position the cursor before editing.

Saving and Exiting

Nano uses `Ctrl` commands for saving and exiting: - **Ctrl+O**: Save (write) the file - **Ctrl+X**: Exit Nano (prompts to save if changes were made) - **Ctrl+O, Enter**: Save without exiting - **Ctrl+O, filename, Enter**: Save as a new file

If prompted to save, press `Y` (yes), `N` (no), or `C` (cancel).

Search and Replace

Searching

- **Ctrl+W**: Search for a string
- **Ctrl+**: Search and replace
- **Alt+W**: Repeat last search (move to next match)
- **Alt+Q**: Repeat last search backward

Example: Press `Ctrl+W`, type `hello`, and press `Enter` to find "hello". Use `Alt+W` to jump to the next occurrence.

Replacing

- **Ctrl+**: Start search and replace
- Enter the search term, press `Enter`
- Enter the replacement term, press `Enter`
- Choose:
  - `Y`: Replace this occurrence
  - `A`: Replace all occurrences
  - `N`: Skip this occurrence
  - `^C`: Cancel

Example: `Ctrl+\`, type `foo`, `Enter`, type `bar`, `Enter`, then `A` to replace all "foo" with "bar".

Copy, Cut, and Paste

Nano handles copy, cut, and paste as follows: - **Ctrl+K**: Cut the current line - **Ctrl+U**: Paste the cut text - **Alt+6**: Copy the current line (if supported) - **Alt+^** (Alt+Shift+6): Copy selected text (after marking) - **Ctrl+6**: Start marking text (select mode), move cursor to select, then use `Alt+^` to copy or `Ctrl+K` to cut

Note: To select text, press `Ctrl+6`, move the cursor to highlight, then cut or copy.

Undo and Redo

- **Alt+U**: Undo last action
- **Alt+E**: Redo undone action

Note: Undo/redo support depends on the Nano version. Older versions may not support this.

Advanced Features

Multiple Files

- Open multiple files: `nano file1 file2`
- Switch files: `Ctrl+Right` (next), `Ctrl+Left` (previous)
- List open buffers: `Alt+,` (if supported)
- Open another file: `Ctrl+R`, type filename, `Enter`

Syntax Highlighting

- Enable: `nano -Y language` (e.g., `nano -Y python file.py`)
- Supported languages include `python`, `c`, `html`, `sh`, etc.

Executing Commands

- **Ctrl+T**: Execute a shell command (e.g., `ls`, `wc`)
- Example: `Ctrl+T`, type `wc %`, `Enter` to count words in the current file.

Spell Checking

- **Ctrl+T, T**: Run spell check (requires `spell` or `aspell` installed)
- Follow prompts to correct misspellings.

Customizing Nano

Nano can be customized via the `~/.nanorc` file. Create or edit it to set preferences.

Example `.nanorc`:

```
set autoindent    # Enable auto-indentation
set tabsize 4     # Set tab width to 4 spaces
set number        # Show line numbers
set mouse         # Enable mouse support
set smooth        # Smooth scrolling
include "/usr/share/nano/*.nanorc"  # Enable syntax highlighting
```

To apply settings in a session: - `nano -l`: Show line numbers - `nano -i`: Enable auto-indent

Tips and Tricks

- **Jump to end of file**: `Alt+/`
- **Jump to start of file**: `Alt+\`
- **Toggle line numbers**: `Alt+N`
- **Enable mouse support**: `Alt+M` (click to move cursor, if enabled)
- **Format paragraphs**: `Ctrl+J` (justifies text)
- **Check Nano version**: `Ctrl+G`, look at the help screen
- **Insert file content**: `Ctrl+R`, type filename, `Enter`
- **Backup files**: `nano -B` creates backup files with `~` suffix

Common Issues and Solutions

1. **Commands Not Working**:
    - Check if you're using `Ctrl` or `Alt` correctly. Some terminals require `Esc` instead of `Alt` (e.g., `Esc, N` for `Alt+N`).
2. **Permission Denied on Save**:

- Use `sudo nano filename` or save to a new file with `Ctrl+O`.

3. **No Syntax Highlighting**:
   - Ensure `*.nanorc` files are included in `~/.nanorc` or use `nano -Y language`.
4. **Lost Changes**:
   - Check for `filename.save` files in the directory. Open with `nano filename.save`.
5. **Slow Performance with Large Files**:
   - Disable syntax highlighting (`nano -A`) or use Vi/Vim for large files.

Conclusion

Nano is an excellent choice for quick edits and users new to Linux text editors. Its simplicity and on-screen shortcuts make it accessible, while features like search-and-replace and syntax highlighting cater to advanced users. For more complex editing, consider exploring Vi or Vim.

For further learning: - Run `nano --help` for command-line options. - Check the Nano man page: `man nano`. - Visit the Nano website or community forums for additional resources.

Vi

Vi: The Original Visual Editor

Vi (Visual Interface) is the original screen-oriented text editor for Unix systems. Created by Bill Joy in 1976, it remains one of the most fundamental and ubiquitous editors in the Unix/Linux world.

Why Learn Vi?

- **Universal availability**: Found on every Linux distribution and Unix system
- **Minimal resource usage**: Works even on minimal Linux installations and embedded systems
- **Foundation knowledge**: Understanding vi helps with vim, neovim, and other vi-like editors
- **Emergency situations**: Often the only editor available in Linux rescue modes and single-user mode
- **System administration**: Essential for Linux server management and configuration
- **POSIX compliance**: Standardized behavior across Linux distributions

Vi Modes

Vi operates in different modes, which is its defining characteristic:

1. Command Mode (Normal Mode)

- Default mode when vi starts
- Used for navigation and text manipulation commands
- Press `Esc` to return to this mode

2. Insert Mode

- Used for typing text
- Enter with `i`, `a`, `o`, or other insert commands
- Exit with `Esc`

3. Command-Line Mode

- Used for file operations and advanced commands
- Enter with `:` from command mode
- Execute with `Enter`

Basic Operations

Starting Vi

```
# Open new file
vi filename.txt

# Open existing file
vi /path/to/file.txt

# Open file at specific line
vi +25 filename.txt

# Open file at first occurrence of pattern
vi +/pattern filename.txt

# Open multiple files
vi file1.txt file2.txt file3.txt
```

Exiting Vi

```
# Save and quit
:wq

# Quit without saving
:q!

# Save file
:w

# Save as new filename
:w newfilename.txt

# Quit (only if no changes)
:q
```

Navigation Commands

Basic Movement

```
# Character movement
h    # Move left
j    # Move down
k    # Move up
l    # Move right

# Word movement
w    # Move to beginning of next word
b    # Move to beginning of previous word
e    # Move to end of current word

# Line movement
0    # Move to beginning of line
^    # Move to first non-blank character
$    # Move to end of line

# Screen movement
H    # Move to top of screen
M    # Move to middle of screen
L    # Move to bottom of screen
```

Advanced Navigation

```
# Page movement
Ctrl+f   # Page forward
Ctrl+b   # Page backward
Ctrl+d   # Half page down
Ctrl+u   # Half page up

# Line jumping
G        # Go to last line
1G       # Go to first line
25G      # Go to line 25
gg       # Go to first line (vim extension)

# Pattern searching
/pattern    # Search forward for pattern
?pattern    # Search backward for pattern
n           # Next search result
N           # Previous search result
```

```
# Character finding
fx       # Find next 'x' on current line
Fx       # Find previous 'x' on current line
tx       # Move to character before next 'x'
Tx       # Move to character after previous 'x'
;        # Repeat last f, F, t, or T command
,        # Repeat last f, F, t, or T command in reverse
```

Text Editing Commands

Inserting Text

```
i    # Insert before cursor
a    # Insert after cursor
I    # Insert at beginning of line
A    # Insert at end of line
o    # Open new line below current line
O    # Open new line above current line
```

Deleting Text

```
x    # Delete character under cursor
X    # Delete character before cursor
dd   # Delete entire line
D    # Delete from cursor to end of line
dw   # Delete word
db   # Delete word backward
d$   # Delete to end of line
d0   # Delete to beginning of line
```

Changing Text

```
r    # Replace single character
R    # Replace mode (overwrite)
cw   # Change word
cc   # Change entire line
C    # Change from cursor to end of line
s    # Substitute character
S    # Substitute line
```

Copying and Pasting

```
yy   # Yank (copy) entire line
Y    # Yank entire line
yw   # Yank word
y$   # Yank to end of line
p    # Paste after cursor
P    # Paste before cursor
```

Working with Multiple Files

```
# Open next file
:n

# Open previous file
:prev

# List all files
:args

# Edit new file
:e filename.txt

# Switch between files
:b filename
```

Search and Replace

Basic Search

```
# Search forward
/search_term

# Search backward
?search_term

# Case-sensitive search
/\Csearch_term

# Whole word search
/\<word\>
```

Search and Replace

```
# Replace first occurrence on current line
:s/old/new/

# Replace all occurrences on current line
:s/old/new/g

# Replace all occurrences in file
:%s/old/new/g

# Replace with confirmation
:%s/old/new/gc

# Replace in specific line range
:1,10s/old/new/g
```

Advanced Features

Marks and Bookmarks

```
# Set mark 'a' at current position
ma

# Go to mark 'a'
'a

# Go to exact position of mark 'a'
`a

# List all marks
:marks
```

Registers

```
# Yank to register 'a'
"ayy

# Paste from register 'a'
"ap

# View all registers
:reg
```

Macros

```
# Start recording macro in register 'a'
qa

# Stop recording
q

# Execute macro 'a'
@a

# Execute macro 'a' 10 times
10@a
```

Configuration

Vi Configuration File (.exrc)

```
# ~/.exrc
set number          # Show line numbers
set autoindent      # Auto-indent new lines
set tabstop=4       # Set tab width
set shiftwidth=4    # Set indent width
set showmatch       # Show matching brackets
set ignorecase      # Ignore case in searches
set wrapscan        # Wrap searches around file
```

Environment Variables

```
# Set default editor
export EDITOR=vi
export VISUAL=vi

# Set vi options
export EXINIT='set number autoindent tabstop=4'
```

Real-World Scenarios

Scenario 1: Emergency Linux System Recovery

```
# Linux system in single-user mode, only vi available
# Edit critical configuration file
vi /etc/fstab

# Navigate to problematic line causing boot failure
/UUID=bad-uuid

# Delete the problematic line
dd

# Save and exit
:wq

# Alternative: Fix systemd service file
vi /etc/systemd/system/problematic.service

# Comment out problematic line
I# <Esc>

# Save and exit
:wq

# Reload systemd and reboot
:!systemctl daemon-reload && reboot
```

Scenario 2: Linux Log File Analysis

```
# Open systemd journal or log file
vi /var/log/syslog
# or
vi /var/log/systemd/journal.log

# Go to end of file to see recent entries
G

# Search backward for critical errors
?CRITICAL
?ERROR
?Failed

# Navigate through error occurrences
```

```
N    # Previous error
n    # Next error

# Copy error line for analysis
yy

# Open new file to save errors
:e /tmp/error_analysis.txt

# Paste error
p

# Add timestamp and system info
O
# Type analysis header
System: $(hostname) - $(date)
Error Analysis:
<Esc>

# Save and return to log
:w
:b syslog
```

Scenario 3: Linux System Configuration

```
# Open multiple Linux configuration files
vi /etc/hosts /etc/resolv.conf /etc/hostname

# Edit first file (/etc/hosts)
# Add new host entry for local network
o
192.168.1.100    newserver.local
192.168.1.101    database.local
192.168.1.102    webserver.local

# Save and move to next file
:w
:n

# Edit resolv.conf for DNS configuration
# Change nameserver to Google DNS
/nameserver
cw
nameserver 8.8.8.8<Esc>
o
```

```
nameserver 8.8.4.4

# Save and move to next file
:w
:n

# Edit hostname for system identification
# Replace entire content
:%d
i
production-server

# Save all and exit
:wq

# Verify changes took effect
:!hostnamectl status
```

Vi vs Modern Editors

Advantages of Vi

- **Ubiquity**: Available everywhere
- **Speed**: Fast startup and operation
- **Keyboard-driven**: No mouse required
- **Powerful**: Complex operations with few keystrokes
- **Stable**: Unchanged interface for decades

Limitations

- **Learning curve**: Modal editing is initially confusing
- **Limited features**: No syntax highlighting, plugins, etc.
- **No GUI**: Text-only interface
- **Basic functionality**: Lacks modern IDE features

Tips and Best Practices

Essential Tips

1. **Master the modes**: Understanding when you're in which mode is crucial
2. **Use Esc liberally**: When in doubt, press Esc to return to command mode
3. **Learn movement commands**: Efficient navigation is key to vi mastery
4. **Practice regularly**: Vi skills deteriorate without use
5. **Use :help**: Built-in help system is comprehensive

Common Mistakes

1. **Typing in command mode**: Results in unintended commands
2. **Forgetting to save**: Always :w before :q
3. **Case sensitivity**: Commands are case-sensitive
4. **Not using counts**: Numbers multiply commands (5dd deletes 5 lines)

Productivity Shortcuts

```
# Repeat last command
.

# Undo last change
u

# Join lines
J

# Change case
~

# Increment number
Ctrl+a

# Decrement number
Ctrl+x
```

Troubleshooting

Common Issues

1. **Can't exit vi**

   ```
   # Press Esc first, then
   :q!
   ```

2. **Accidentally in replace mode**

   ```
   # Press Esc to return to command mode
   Esc
   ```

3. **Lost changes**

   ```
   # Check for swap file
   :recover
   ```

4. **File is read-only**

```
# Force write
:w!
# Or save as new file
:w newfilename
```

Recovery Options

```
# Recover from swap file
vi -r filename.txt

# List available swap files
vi -r

# Start vi in recovery mode
vi -r
```

Vi remains an essential skill for any Linux user or system administrator. While modern editors offer more features, vi's simplicity, universality, and efficiency make it invaluable for quick edits, system recovery, and situations where other editors aren't available.

Vim

Vim: Vi Improved

Vim (Vi Improved) is an enhanced version of the vi editor, created by Bram Moolenaar in 1991. It extends vi with numerous features while maintaining backward compatibility, making it one of the most popular and powerful text editors for developers.

Why Choose Vim?

- **Powerful and extensible**: Thousands of plugins and customizations
- **Efficient editing**: Modal editing with powerful commands
- **Native Linux integration**: Seamlessly integrates with Linux workflows
- **Active community**: Large ecosystem of plugins and configurations
- **Programming-focused**: Excellent support for Linux development workflows
- **Highly customizable**: Extensive configuration options
- **System administration**: Perfect for Linux server management and configuration

Installation

Linux Distributions

```
# Ubuntu/Debian
sudo apt update
sudo apt install vim

# CentOS/RHEL/Fedora
sudo yum install vim
# or
sudo dnf install vim

# Arch Linux
sudo pacman -S vim

# Alpine Linux
sudo apk add vim
```

Building from Source

```
# Clone Vim repository
git clone https://github.com/vim/vim.git
cd vim

# Configure build
./configure --with-features=huge \
            --enable-multibyte \
            --enable-python3interp \
            --enable-gui=gtk3 \
            --enable-cscope

# Compile and install
make
sudo make install
```

Vim Modes

Vim extends vi's modal concept with additional modes:

1. Normal Mode

- Default mode for navigation and commands
- Press `Esc` to enter from any other mode

2. Insert Mode

- For typing text
- Enter with `i`, `a`, `o`, `I`, `A`, `O`

3. Visual Mode

- For selecting text
- `v` for character-wise selection
- `V` for line-wise selection
- `Ctrl+v` for block-wise selection

4. Command-Line Mode

- For ex commands
- Enter with `:`, `/`, `?`

5. Replace Mode

- For overwriting text
- Enter with `R`

Enhanced Navigation

Advanced Movement

```
# Word movement (enhanced)
w    # Next word start
W    # Next WORD start (whitespace-separated)
b    # Previous word start
B    # Previous WORD start
e    # End of word
E    # End of WORD
ge   # End of previous word

# Paragraph movement
{    # Previous paragraph
}    # Next paragraph

# Sentence movement
(    # Previous sentence
)    # Next sentence

# Function movement (programming)
[[   # Previous function
]]   # Next function
[{   # Previous unmatched {
]}   # Next unmatched }

# Jump list navigation
Ctrl+o   # Jump to previous location
Ctrl+i   # Jump to next location
```

Window and Tab Management

```
# Window splitting
:split filename    # Horizontal split
:vsplit filename   # Vertical split
Ctrl+w s          # Split current window horizontally
Ctrl+w v          # Split current window vertically
```

```
# Window navigation
Ctrl+w h          # Move to left window
Ctrl+w j          # Move to bottom window
Ctrl+w k          # Move to top window
Ctrl+w l          # Move to right window
Ctrl+w w          # Cycle through windows

# Window resizing
Ctrl+w +          # Increase height
Ctrl+w -          # Decrease height
Ctrl+w >          # Increase width
Ctrl+w <          # Decrease width
Ctrl+w =          # Equalize window sizes

# Tab management
:tabnew filename  # Open new tab
:tabclose         # Close current tab
gt                # Next tab
gT                # Previous tab
:tabs             # List all tabs
```

Advanced Text Editing

Visual Mode Operations

```
# Select text and perform operations
v                 # Start visual selection
V                 # Select entire lines
Ctrl+v           # Block selection

# Operations on selected text
d                # Delete selection
y                # Yank (copy) selection
c                # Change selection
>                # Indent selection
<                # Unindent selection
=                # Auto-indent selection
```

Advanced Copy/Paste

```
# Named registers
"ayy              # Yank line to register 'a'
"ap               # Paste from register 'a'
"Ayy              # Append line to register 'a'

# Special registers
"+y               # Yank to system clipboard
"+p               # Paste from system clipboard
"*y               # Yank to selection clipboard
"*p               # Paste from selection clipboard
""p               # Paste from unnamed register
".p               # Paste last inserted text
```

Text Objects

```
# Inner text objects
iw                # Inner word
is                # Inner sentence
ip                # Inner paragraph
it                # Inner tag
i"                # Inner quotes
i(                # Inner parentheses
i{                # Inner braces
i[                # Inner brackets

# Around text objects
aw                # Around word (includes spaces)
as                # Around sentence
ap                # Around paragraph
at                # Around tag
a"                # Around quotes
a(                # Around parentheses
a{                # Around braces
a[                # Around brackets

# Usage examples
diw               # Delete inner word
ci"               # Change inner quotes
yap               # Yank around paragraph
```

Configuration and Customization

Vimrc Configuration

```
" ~/.vimrc - Basic configuration

" General settings
set nocompatible              " Disable vi compatibility
set number                    " Show line numbers
set relativenumber           " Show relative line numbers
set ruler                     " Show cursor position
set showcmd                   " Show command in status line
set showmatch                " Show matching brackets
set hlsearch                 " Highlight search results
set incsearch                " Incremental search
set ignorecase               " Ignore case in search
set smartcase                " Smart case sensitivity
set autoindent               " Auto-indent new lines
set smartindent              " Smart indentation
set expandtab                " Use spaces instead of tabs
set tabstop=4                " Tab width
set shiftwidth=4             " Indent width
set softtabstop=4            " Soft tab width
set wrap                     " Wrap long lines
set linebreak                " Break lines at word boundaries
set scrolloff=5              " Keep 5 lines visible around cursor
set sidescrolloff=5          " Keep 5 columns visible around cursor
set backspace=indent,eol,start " Allow backspace over everything
set wildmenu                 " Enhanced command completion
set wildmode=longest:full,full " Command completion mode
set laststatus=2             " Always show status line
set encoding=utf-8           " Use UTF-8 encoding
set fileencoding=utf-8       " File encoding
set termencoding=utf-8       " Terminal encoding

" Visual settings
syntax enable                " Enable syntax highlighting
set background=dark          " Dark background
colorscheme desert           " Color scheme
set cursorline              " Highlight current line
set colorcolumn=80          " Show column at 80 characters

" Search settings
set path+=**                " Search in subdirectories
set wildignore+=*.o,*.obj,*.pyc,*.class " Ignore compiled files
```

```
" Backup and swap settings
set backup                  " Enable backups
set backupdir=~/.vim/backup " Backup directory
set directory=~/.vim/swap   " Swap file directory
set undofile                " Persistent undo
set undodir=~/.vim/undo     " Undo directory

" Create directories if they don't exist
if !isdirectory($HOME."/.vim/backup")
    call mkdir($HOME."/.vim/backup", "p")
endif
if !isdirectory($HOME."/.vim/swap")
    call mkdir($HOME."/.vim/swap", "p")
endif
if !isdirectory($HOME."/.vim/undo")
    call mkdir($HOME."/.vim/undo", "p")
endif

" Key mappings
let mapleader = ","         " Set leader key
nnoremap <leader>w :w<CR>   " Quick save
nnoremap <leader>q :q<CR>   " Quick quit
nnoremap <leader>x :x<CR>   " Quick save and quit
nnoremap <leader>h :noh<CR> " Clear search highlighting
nnoremap <C-h> <C-w>h       " Window navigation
nnoremap <C-j> <C-w>j
nnoremap <C-k> <C-w>k
nnoremap <C-l> <C-w>l

" Function key mappings
nnoremap <F2> :set number!<CR>     " Toggle line numbers
nnoremap <F3> :set paste!<CR>      " Toggle paste mode
nnoremap <F4> :set wrap!<CR>       " Toggle line wrapping
nnoremap <F5> :source ~/.vimrc<CR> " Reload vimrc

" Auto commands
autocmd BufWritePre * :%s/\s\+$//e " Remove trailing whitespace on save
autocmd BufRead,BufNewFile *.py set filetype=python
autocmd BufRead,BufNewFile *.js set filetype=javascript
autocmd BufRead,BufNewFile *.html set filetype=html
```

Advanced Vimrc Features

```
" Advanced ~/.vimrc additions

" Status line customization
set statusline=%f         " File name
set statusline+=%m        " Modified flag
set statusline+=%r        " Read-only flag
set statusline+=%h        " Help flag
set statusline+=%w        " Preview flag
set statusline+=%=        " Right align
set statusline+=%y        " File type
set statusline+=[%{&ff}]  " File format
set statusline+=[%{&fenc}] " File encoding
set statusline+=\ %l/%L   " Line number/total lines
set statusline+=\ %c      " Column number
set statusline+=\ %P      " Percentage through file

" Custom functions
function! ToggleBackground()
    if &background == 'dark'
        set background=light
    else
        set background=dark
    endif
endfunction
nnoremap <F6> :call ToggleBackground()<CR>

" Word count function
function! WordCount()
    let s:old_status = v:statusmsg
    let position = getpos(".")
    exe ":silent normal g\<c-g>"
    let stat = v:statusmsg
    let s:word_count = 0
    if stat != '--No lines in buffer--'
        let s:word_count = str2nr(split(v:statusmsg)[11])
        let v:statusmsg = s:old_status
    end
    call setpos('.', position)
    return s:word_count
endfunction

" File type specific settings
autocmd FileType python setlocal expandtab shiftwidth=4 softtabstop=4
autocmd FileType javascript setlocal expandtab shiftwidth=2 softtabstop=2
autocmd FileType html setlocal expandtab shiftwidth=2 softtabstop=2
```

```
autocmd FileType css setlocal expandtab shiftwidth=2 softtabstop=2
autocmd FileType yaml setlocal expandtab shiftwidth=2 softtabstop=2
autocmd FileType markdown setlocal wrap linebreak nolist textwidth=0 wrapmargin=0

" Abbreviations
iabbrev teh the
iabbrev adn and
iabbrev seperate separate
iabbrev recieve receive
```

Plugin Management

Using vim-plug

```
# Install vim-plug
curl -fLo ~/.vim/autoload/plug.vim --create-dirs \
    https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim
```

```
" Add to ~/.vimrc
call plug#begin('~/.vim/plugged')

" Essential plugins
Plug 'tpope/vim-sensible'           " Sensible defaults
Plug 'tpope/vim-surround'           " Surround text objects
Plug 'tpope/vim-commentary'         " Comment/uncomment
Plug 'tpope/vim-fugitive'           " Git integration
Plug 'scrooloose/nerdtree'          " File explorer
Plug 'ctrlpvim/ctrlp.vim'           " Fuzzy file finder
Plug 'vim-airline/vim-airline'      " Status line
Plug 'vim-airline/vim-airline-themes' " Airline themes
Plug 'airblade/vim-gitgutter'       " Git diff in gutter
Plug 'dense-analysis/ale'           " Linting and fixing
Plug 'ycm-core/YouCompleteMe'       " Code completion
Plug 'preservim/tagbar'             " Tag browser
Plug 'junegunn/fzf', { 'do': { -> fzf#install() } }
Plug 'junegunn/fzf.vim'             " FZF integration

" Language-specific plugins
Plug 'vim-python/python-syntax'     " Python syntax
Plug 'pangloss/vim-javascript'      " JavaScript syntax
Plug 'mxw/vim-jsx'                  " JSX syntax
Plug 'rust-lang/rust.vim'          " Rust support
Plug 'fatih/vim-go'                 " Go support
```

```
" Color schemes
Plug 'morhetz/gruvbox'               " Gruvbox theme
Plug 'dracula/vim', { 'as': 'dracula' } " Dracula theme
Plug 'joshdick/onedark.vim'          " One Dark theme

call plug#end()

" Plugin configurations
let g:airline_theme='gruvbox'
let g:gruvbox_contrast_dark='hard'
colorscheme gruvbox

" NERDTree configuration
nnoremap <F7> :NERDTreeToggle<CR>
let NERDTreeShowHidden=1

" CtrlP configuration
let g:ctrlp_map = '<c-p>'
let g:ctrlp_cmd = 'CtrlP'
let g:ctrlp_working_path_mode = 'ra'

" ALE configuration
let g:ale_linters = {
\   'python': ['flake8', 'pylint'],
\   'javascript': ['eslint'],
\}
let g:ale_fixers = {
\   'python': ['autopep8', 'black'],
\   'javascript': ['prettier'],
\}
```

Installing Plugins

```
" In Vim command mode
:PlugInstall    " Install plugins
:PlugUpdate     " Update plugins
:PlugClean      " Remove unused plugins
:PlugStatus     " Check plugin status
```

Programming with Vim

Code Navigation

```
# Tag navigation (requires ctags)
Ctrl+]          # Jump to tag definition
Ctrl+t          # Jump back from tag
:tag function   # Jump to specific tag
:tags           # Show tag stack

# Code folding
zf              # Create fold
zo              # Open fold
zc              # Close fold
za              # Toggle fold
zR              # Open all folds
zM              # Close all folds
```

Code Completion

```
# Built-in completion
Ctrl+n          # Next completion
Ctrl+p          # Previous completion
Ctrl+x Ctrl+f   # File name completion
Ctrl+x Ctrl+l   # Line completion
Ctrl+x Ctrl+k   # Dictionary completion
Ctrl+x Ctrl+o   # Omni completion
```

Debugging Integration

```
" GDB integration
packadd termdebug
:Termdebug program_name

" Debugging commands
:Break          " Set breakpoint
:Clear          " Clear breakpoint
:Step           " Step into
:Over           # Step over
:Continue       " Continue execution
```

Real-World Development Scenarios

Scenario 1: Python Development Setup

```
" Python-specific vimrc additions
autocmd FileType python setlocal expandtab shiftwidth=4 softtabstop=4
autocmd FileType python setlocal textwidth=79
autocmd FileType python setlocal colorcolumn=80
autocmd FileType python nnoremap <buffer> <F9> :exec '!python' shellescape(@%, 1)<cr>

" Python debugging
autocmd FileType python nnoremap <buffer> <F10> :exec '!python -m pdb' shellescape(@%, 1)<cr

" Virtual environment detection
py3 << EOF
import os
import sys
if 'VIRTUAL_ENV' in os.environ:
    project_base_dir = os.environ['VIRTUAL_ENV']
    activate_this = os.path.join(project_base_dir, 'bin/activate_this.py')
    exec(open(activate_this).read(), dict(__file__=activate_this))
EOF
```

Scenario 2: Web Development Workflow

```
" Web development configuration
autocmd FileType html,css,javascript setlocal expandtab shiftwidth=2 softtabstop=2

" Live reload setup
autocmd BufWritePost *.html,*.css,*.js silent !browser-sync reload

" Emmet integration
let g:user_emmet_leader_key=','

" Auto-close tags
let g:closetag_filenames = '*.html,*.xhtml,*.phtml,*.jsx'

" Prettier formatting
autocmd BufWritePre *.js,*.jsx,*.css,*.html PrettierAsync
```

Scenario 3: Linux System Administration

```
" Linux configuration file editing
autocmd BufRead,BufNewFile /etc/* setlocal nobackup nowritebackup
autocmd BufRead,BufNewFile /etc/systemd/system/* setlocal filetype=systemd
autocmd BufRead,BufNewFile *.conf setlocal syntax=conf
autocmd BufRead,BufNewFile /etc/nginx/* setlocal filetype=nginx
autocmd BufRead,BufNewFile /etc/apache2/* setlocal filetype=apache

" Linux log file viewing
autocmd BufRead,BufNewFile /var/log/* setlocal readonly
autocmd BufRead,BufNewFile /var/log/* normal G
autocmd BufRead,BufNewFile /var/log/syslog setlocal filetype=messages
autocmd BufRead,BufNewFile /var/log/auth.log setlocal filetype=messages

" Sudo save function for system files
command! W w !sudo tee % > /dev/null
command! Wsudo w !sudo tee % > /dev/null

" Quick Linux config editing
nnoremap <leader>ev :vsplit $MYVIMRC<cr>
nnoremap <leader>sv :source $MYVIMRC<cr>
nnoremap <leader>eh :vsplit /etc/hosts<cr>
nnoremap <leader>es :vsplit /etc/ssh/sshd_config<cr>
nnoremap <leader>en :vsplit /etc/nginx/nginx.conf<cr>

" Systemd service file templates
autocmd BufNewFile *.service 0r ~/.vim/templates/systemd.service
autocmd BufNewFile *.timer 0r ~/.vim/templates/systemd.timer

" Linux-specific commands
command! Systemctl !sudo systemctl
command! Journalctl !journalctl
command! Dmesg !dmesg | tail -20
```

Advanced Vim Techniques

Macros and Automation

```
# Recording macros
qa              # Start recording macro 'a'
# ... perform actions ...
q               # Stop recording
```

```
# Playing macros
@a              # Execute macro 'a'
@@              # Repeat last macro
10@a            # Execute macro 'a' 10 times

# Editing macros
:let @a='...'   # Edit macro 'a' directly
```

Regular Expressions

```
# Search patterns
/\v<word>       # Very magic mode - exact word
/\c<Word>       # Case insensitive search
/\C<Word>       # Case sensitive search

# Substitution patterns
:%s/\v(\w+)\s+(\w+)/\2 \1/g    # Swap words
:%s/\v^(.{80}).*/\1/g          # Truncate lines to 80 chars
:%s/\v\s+$//g                  # Remove trailing whitespace
```

Custom Commands

```
" Custom commands in vimrc
command! -nargs=1 Search execute 'vimgrep /<args>/gj **/*.py' | copen
command! Todo execute 'vimgrep /TODO\|FIXME\|XXX/gj **/*' | copen
command! RemoveTrailingSpaces %s/\s\+$//e
command! -range=% FormatJSON <line1>,<line2>!python -m json.tool

" Usage
:Search function_name
:Todo
:RemoveTrailingSpaces
:FormatJSON
```

Performance Optimization

Vim Performance Tips

```
" Performance optimizations
set lazyredraw              " Don't redraw during macros
set ttyfast                 " Fast terminal connection
set synmaxcol=200          " Limit syntax highlighting column
```

```
set regexpengine=1         " Use old regex engine for speed

" Disable unused features
set noshowmatch            " Don't show matching brackets
let loaded_matchparen = 1  " Disable matchparen plugin
```

Large File Handling

```
" Large file handling
autocmd BufReadPre * if getfsize(expand("%")) > 10000000 | syntax off | endif

" Function for large files
function! LargeFile()
    syntax off
    set eventignore+=FileType
    setlocal bufhidden=unload
    setlocal buftype=nowrite
    setlocal undolevels=-1
endfunction
autocmd BufReadPre * if getfsize(expand("%")) > 50000000 | call LargeFile() | endif
```

Troubleshooting Common Issues

Performance Problems

```
# Profile Vim startup
vim --startuptime startup.log

# Check what's slowing down Vim
:profile start profile.log
:profile func *
:profile file *
# ... use Vim normally ...
:profile pause
:noautocmd qall!
```

Plugin Conflicts

```
# Start Vim without plugins
vim -u NONE -N
```

```
# Start with minimal config
vim -u ~/.vimrc.minimal

# Debug plugin loading
vim -V9myVim.log
```

Recovery and Backup

```
# Recover from swap file
vim -r filename.txt

# List swap files
vim -r

# Disable swap files (not recommended)
set noswapfile
```

Vim's power lies in its extensibility and efficiency. While it has a steep learning curve, mastering Vim can significantly improve your editing speed and workflow, especially for programming and system administration tasks.

Neovim

Neovim: The Future of Vim

Neovim is a modern fork of Vim that focuses on extensibility, usability, and performance. Created in 2014, it maintains Vim compatibility while adding powerful new features like built-in LSP support, Lua scripting, and better plugin architecture.

Why Choose Neovim?

- **Modern architecture**: Asynchronous job control and better plugin API
- **Built-in LSP**: Native Language Server Protocol support for Linux development
- **Lua scripting**: More powerful and faster than Vimscript
- **Better defaults**: Sensible configuration out of the box
- **Active development**: Regular updates and new features
- **Tree-sitter**: Advanced syntax highlighting and code parsing
- **Embedded terminal**: Built-in terminal emulator perfect for Linux workflows
- **Linux-first design**: Optimized for Linux development environments

Installation

Package Managers

```
# Ubuntu/Debian (latest stable)
sudo apt update
sudo apt install neovim

# Ubuntu/Debian (latest unstable)
sudo add-apt-repository ppa:neovim-ppa/unstable
sudo apt update
sudo apt install neovim

# CentOS/RHEL/Fedora
sudo dnf install neovim

# Arch Linux
sudo pacman -S neovim

# SUSE/openSUSE
sudo zypper install neovim
```

```
# Gentoo
sudo emerge app-editors/neovim
```

From Source

```
# Install dependencies (Ubuntu/Debian)
sudo apt install ninja-build gettext libtool libtool-bin autoconf automake cmake g++ pkg-con

# Clone and build
git clone https://github.com/neovim/neovim
cd neovim
make CMAKE_BUILD_TYPE=RelWithDebInfo
sudo make install
```

AppImage (Portable)

```
# Download AppImage
curl -LO https://github.com/neovim/neovim/releases/latest/download/nvim.appimage
chmod u+x nvim.appimage
sudo mv nvim.appimage /usr/local/bin/nvim
```

Configuration Structure

Neovim uses a different configuration structure than Vim:

```
# Configuration directory
~/.config/nvim/

# Main configuration file
~/.config/nvim/init.lua    # Lua configuration (recommended)
# or
~/.config/nvim/init.vim    # Vimscript configuration

# Plugin directory
~/.local/share/nvim/site/pack/
```

Basic Lua Configuration

init.lua Structure

```lua
-- ~/.config/nvim/init.lua

-- Basic settings
vim.opt.number = true
vim.opt.relativenumber = true
vim.opt.expandtab = true
vim.opt.shiftwidth = 4
vim.opt.tabstop = 4
vim.opt.softtabstop = 4
vim.opt.smartindent = true
vim.opt.wrap = false
vim.opt.swapfile = false
vim.opt.backup = false
vim.opt.undodir = os.getenv("HOME") .. "/.vim/undodir"
vim.opt.undofile = true
vim.opt.hlsearch = false
vim.opt.incsearch = true
vim.opt.termguicolors = true
vim.opt.scrolloff = 8
vim.opt.signcolumn = "yes"
vim.opt.isfname:append("@-@")
vim.opt.updatetime = 50
vim.opt.colorcolumn = "80"

-- Leader key
vim.g.mapleader = " "

-- Key mappings
vim.keymap.set("n", "<leader>pv", vim.cmd.Ex)
vim.keymap.set("v", "J", ":m '>+1<CR>gv=gv")
vim.keymap.set("v", "K", ":m '<-2<CR>gv=gv")
vim.keymap.set("n", "J", "mzJ`z")
vim.keymap.set("n", "<C-d>", "<C-d>zz")
vim.keymap.set("n", "<C-u>", "<C-u>zz")
vim.keymap.set("n", "n", "nzzzv")
vim.keymap.set("n", "N", "Nzzzv")

-- System clipboard
vim.keymap.set({"n", "v"}, "<leader>y", [["+y]])
vim.keymap.set("n", "<leader>Y", [["+Y]])
vim.keymap.set({"n", "v"}, "<leader>d", [["_d]])
```

```lua
-- Replace word under cursor
vim.keymap.set("n", "<leader>s", [[:%s/\<<C-r><C-w>\>/<C-r><C-w>/gI<Left><Left><Left>]])

-- Make file executable
vim.keymap.set("n", "<leader>x", "<cmd>!chmod +x %<CR>", { silent = true })

-- Auto commands
local augroup = vim.api.nvim_create_augroup
local autocmd = vim.api.nvim_create_autocmd

local yank_group = augroup('HighlightYank', {})
autocmd('TextYankPost', {
    group = yank_group,
    pattern = '*',
    callback = function()
        vim.highlight.on_yank({
            higroup = 'IncSearch',
            timeout = 40,
        })
    end,
})

-- Remove trailing whitespace on save
local trim_group = augroup('TrimWhitespace', {})
autocmd('BufWritePre', {
    group = trim_group,
    pattern = '*',
    command = '%s/\\s\\+$//e',
})
```

Plugin Management with Packer

Installing Packer

```
git clone --depth 1 https://github.com/wbthomason/packer.nvim\
 ~/.local/share/nvim/site/pack/packer/start/packer.nvim
```

Packer Configuration

```lua
-- ~/.config/nvim/lua/plugins.lua

-- This file can be loaded by calling `lua require('plugins')` from your init.lua
```

```
-- Only required if you have packer configured as `opt`
vim.cmd [[packadd packer.nvim]]

return require('packer').startup(function(use)
  -- Packer can manage itself
  use 'wbthomason/packer.nvim'

  -- Fuzzy finder
  use {
    'nvim-telescope/telescope.nvim', tag = '0.1.4',
    requires = { {'nvim-lua/plenary.nvim'} }
  }

  -- Color schemes
  use({
    'rose-pine/neovim',
    as = 'rose-pine',
    config = function()
      vim.cmd('colorscheme rose-pine')
    end
  })

  use('nvim-treesitter/nvim-treesitter', {run = ':TSUpdate'})
  use('nvim-treesitter/playground')
  use('theprimeagen/harpoon')
  use('mbbill/undotree')
  use('tpope/vim-fugitive')

  -- LSP Support
  use {
    'VonHeikemen/lsp-zero.nvim',
    branch = 'v3.x',
    requires = {
      --- Uncomment these if you want to manage LSP servers from neovim
      {'williamboman/mason.nvim'},
      {'williamboman/mason-lspconfig.nvim'},

      -- LSP Support
      {'neovim/nvim-lspconfig'},
      -- Autocompletion
      {'hrsh7th/nvim-cmp'},
      {'hrsh7th/cmp-nvim-lsp'},
      {'L3MON4D3/LuaSnip'},
    }
  }

  -- File explorer
```

```
use {
  'nvim-tree/nvim-tree.lua',
  requires = {
    'nvim-tree/nvim-web-devicons', -- optional, for file icons
  },
  tag = 'nightly' -- optional, updated every week
}

-- Status line
use {
  'nvim-lualine/lualine.nvim',
  requires = { 'nvim-tree/nvim-web-devicons', opt = true }
}

-- Git integration
use {
  'lewis6991/gitsigns.nvim',
  config = function()
    require('gitsigns').setup()
  end
}

-- Auto pairs
use {
  "windwp/nvim-autopairs",
  config = function() require("nvim-autopairs").setup {} end
}

-- Comments
use {
  'numToStr/Comment.nvim',
  config = function()
    require('Comment').setup()
  end
}

-- Terminal integration
use {"akinsho/toggleterm.nvim", tag = '*', config = function()
  require("toggleterm").setup()
end}

-- Which key
use {
  "folke/which-key.nvim",
  config = function()
    vim.o.timeout = true
    vim.o.timeoutlen = 300
```

```
      require("which-key").setup {}
    end
  }

  -- Indent guides
  use "lukas-reineke/indent-blankline.nvim"

  -- Buffer line
  use {'akinsho/bufferline.nvim', tag = "*", requires = 'nvim-tree/nvim-web-devicons'}

end)
```

Loading Plugins in init.lua

```
-- ~/.config/nvim/init.lua
require('plugins')

-- Plugin configurations
require('telescope').setup{}
require('nvim-tree').setup{}
require('lualine').setup{}
require('bufferline').setup{}
```

Language Server Protocol (LSP) Setup

LSP Configuration

```
-- ~/.config/nvim/lua/lsp-config.lua

local lsp_zero = require('lsp-zero')

lsp_zero.on_attach(function(client, bufnr)
  local opts = {buffer = bufnr, remap = false}

  vim.keymap.set("n", "gd", function() vim.lsp.buf.definition() end, opts)
  vim.keymap.set("n", "K", function() vim.lsp.buf.hover() end, opts)
  vim.keymap.set("n", "<leader>vws", function() vim.lsp.buf.workspace_symbol() end, opts)
  vim.keymap.set("n", "<leader>vd", function() vim.diagnostic.open_float() end, opts)
  vim.keymap.set("n", "[d", function() vim.diagnostic.goto_next() end, opts)
  vim.keymap.set("n", "]d", function() vim.diagnostic.goto_prev() end, opts)
  vim.keymap.set("n", "<leader>vca", function() vim.lsp.buf.code_action() end, opts)
  vim.keymap.set("n", "<leader>vrr", function() vim.lsp.buf.references() end, opts)
  vim.keymap.set("n", "<leader>vrn", function() vim.lsp.buf.rename() end, opts)
```

```
  vim.keymap.set("i", "<C-h>", function() vim.lsp.buf.signature_help() end, opts)
end)

-- Mason setup for LSP server management
require('mason').setup({})
require('mason-lspconfig').setup({
  ensure_installed = {
    'tsserver',
    'rust_analyzer',
    'gopls',
    'pyright',
    'lua_ls',
    'clangd',
  },
  handlers = {
    lsp_zero.default_setup,
    lua_ls = function()
      local lua_opts = lsp_zero.nvim_lua_ls()
      require('lspconfig').lua_ls.setup(lua_opts)
    end,
  }
})

-- Completion setup
local cmp = require('cmp')
local cmp_select = {behavior = cmp.SelectBehavior.Select}

cmp.setup({
  sources = {
    {name = 'path'},
    {name = 'nvim_lsp'},
    {name = 'nvim_lua'},
    {name = 'luasnip', keyword_length = 2},
    {name = 'buffer', keyword_length = 3},
  },
  formatting = lsp_zero.cmp_format(),
  mapping = cmp.mapping.preset.insert({
    ['<C-p>'] = cmp.mapping.select_prev_item(cmp_select),
    ['<C-n>'] = cmp.mapping.select_next_item(cmp_select),
    ['<C-y>'] = cmp.mapping.confirm({ select = true }),
    ['<C-Space>'] = cmp.mapping.complete(),
  }),
})
```

Installing Language Servers

```
# Using Mason (recommended)
:Mason
# Then install servers interactively

# Or install manually
npm install -g typescript-language-server
pip install python-lsp-server
go install golang.org/x/tools/gopls@latest
```

Tree-sitter Configuration

Tree-sitter Setup

```
-- ~/.config/nvim/lua/treesitter-config.lua

require'nvim-treesitter.configs'.setup {
  -- A list of parser names, or "all"
  ensure_installed = {
    "c", "lua", "vim", "vimdoc", "query", "python", "javascript",
    "typescript", "rust", "go", "html", "css", "json", "yaml", "markdown"
  },

  -- Install parsers synchronously (only applied to `ensure_installed`)
  sync_install = false,

  -- Automatically install missing parsers when entering buffer
  auto_install = true,

  highlight = {
    enable = true,
    -- Setting this to true will run `:h syntax` and tree-sitter at the same time.
    additional_vim_regex_highlighting = false,
  },

  indent = {
    enable = true
  },

  incremental_selection = {
    enable = true,
    keymaps = {
      init_selection = "gnn",
```

```
      node_incremental = "grn",
      scope_incremental = "grc",
      node_decremental = "grm",
    },
  },

  textobjects = {
    select = {
      enable = true,
      lookahead = true,
      keymaps = {
        ["af"] = "@function.outer",
        ["if"] = "@function.inner",
        ["ac"] = "@class.outer",
        ["ic"] = "@class.inner",
      },
    },
  },
}
```

Advanced Neovim Features

Telescope Configuration

```
-- ~/.config/nvim/lua/telescope-config.lua

local builtin = require('telescope.builtin')

vim.keymap.set('n', '<leader>pf', builtin.find_files, {})
vim.keymap.set('n', '<C-p>', builtin.git_files, {})
vim.keymap.set('n', '<leader>ps', function()
    builtin.grep_string({ search = vim.fn.input("Grep > ") })
end)
vim.keymap.set('n', '<leader>vh', builtin.help_tags, {})

require('telescope').setup{
  defaults = {
    file_ignore_patterns = {
      "node_modules",
      ".git",
      "dist",
      "build"
    },
    mappings = {
      i = {
```

```
        ["<C-h>"] = "which_key"
      }
    }
  },
  pickers = {
    find_files = {
      theme = "dropdown",
    }
  },
  extensions = {
    fzf = {
      fuzzy = true,
      override_generic_sorter = true,
      override_file_sorter = true,
      case_mode = "smart_case",
    }
  }
}

-- Load extensions
require('telescope').load_extension('fzf')
```

Terminal Integration

```
-- ~/.config/nvim/lua/terminal-config.lua

require("toggleterm").setup{
  size = 20,
  open_mapping = [[<c-\>]],
  hide_numbers = true,
  shade_filetypes = {},
  shade_terminals = true,
  shading_factor = 2,
  start_in_insert = true,
  insert_mappings = true,
  persist_size = true,
  direction = "float",
  close_on_exit = true,
  shell = vim.o.shell,
  float_opts = {
    border = "curved",
    winblend = 0,
    highlights = {
      border = "Normal",
      background = "Normal",
```

```
    },
  },
}

-- Terminal keymaps
function _G.set_terminal_keymaps()
  local opts = {buffer = 0}
  vim.keymap.set('t', '<esc>', [[<C-\><C-n>]], opts)
  vim.keymap.set('t', 'jk', [[<C-\><C-n>]], opts)
  vim.keymap.set('t', '<C-h>', [[<Cmd>wincmd h<CR>]], opts)
  vim.keymap.set('t', '<C-j>', [[<Cmd>wincmd j<CR>]], opts)
  vim.keymap.set('t', '<C-k>', [[<Cmd>wincmd k<CR>]], opts)
  vim.keymap.set('t', '<C-l>', [[<Cmd>wincmd l<CR>]], opts)
end

vim.cmd('autocmd! TermOpen term://* lua set_terminal_keymaps()')
```

Real-World Development Setups

Python Development

```
-- ~/.config/nvim/lua/python-config.lua

-- Python-specific settings
vim.api.nvim_create_autocmd("FileType", {
  pattern = "python",
  callback = function()
    vim.opt_local.expandtab = true
    vim.opt_local.shiftwidth = 4
    vim.opt_local.tabstop = 4
    vim.opt_local.softtabstop = 4
    vim.opt_local.colorcolumn = "88"
  end,
})

-- Python debugging with DAP
local dap = require('dap')
dap.adapters.python = {
  type = 'executable',
  command = 'python',
  args = { '-m', 'debugpy.adapter' },
}

dap.configurations.python = {
```

```
  {
    type = 'python',
    request = 'launch',
    name = "Launch file",
    program = "${file}",
    pythonPath = function()
      return '/usr/bin/python'
    end,
  },
}

-- Python testing
vim.keymap.set('n', '<leader>pt', ':!python -m pytest %<CR>')
vim.keymap.set('n', '<leader>pr', ':!python %<CR>')
```

JavaScript/TypeScript Development

```
-- ~/.config/nvim/lua/js-config.lua

-- JavaScript/TypeScript settings
vim.api.nvim_create_autocmd("FileType", {
  pattern = {"javascript", "typescript", "javascriptreact", "typescriptreact"},
  callback = function()
    vim.opt_local.expandtab = true
    vim.opt_local.shiftwidth = 2
    vim.opt_local.tabstop = 2
    vim.opt_local.softtabstop = 2
  end,
})

-- ESLint and Prettier integration
local null_ls = require("null-ls")
null_ls.setup({
  sources = {
    null_ls.builtins.formatting.prettier,
    null_ls.builtins.diagnostics.eslint,
    null_ls.builtins.code_actions.eslint,
  },
})

-- Auto-format on save
vim.api.nvim_create_autocmd("BufWritePre", {
  pattern = {"*.js", "*.jsx", "*.ts", "*.tsx"},
  callback = function()
    vim.lsp.buf.format()
```

```
  end,
})
```

Go Development

```
-- ~/.config/nvim/lua/go-config.lua

-- Go settings
vim.api.nvim_create_autocmd("FileType", {
  pattern = "go",
  callback = function()
    vim.opt_local.expandtab = false
    vim.opt_local.tabstop = 4
    vim.opt_local.shiftwidth = 4
  end,
})

-- Go-specific keymaps
vim.api.nvim_create_autocmd("FileType", {
  pattern = "go",
  callback = function()
    local opts = { buffer = true }
    vim.keymap.set('n', '<leader>gr', ':!go run %<CR>', opts)
    vim.keymap.set('n', '<leader>gt', ':!go test<CR>', opts)
    vim.keymap.set('n', '<leader>gb', ':!go build<CR>', opts)
    vim.keymap.set('n', '<leader>gf', ':!gofmt -w %<CR>', opts)
  end,
})
```

Debugging with DAP

DAP Configuration

```
-- ~/.config/nvim/lua/dap-config.lua

local dap = require('dap')
local dapui = require('dapui')

-- DAP UI setup
dapui.setup()

-- Auto-open/close DAP UI
dap.listeners.after.event_initialized["dapui_config"] = function()
```

```
  dapui.open()
end
dap.listeners.before.event_terminated["dapui_config"] = function()
  dapui.close()
end
dap.listeners.before.event_exited["dapui_config"] = function()
  dapui.close()
end

-- Keymaps
vim.keymap.set('n', '<F5>', dap.continue)
vim.keymap.set('n', '<F10>', dap.step_over)
vim.keymap.set('n', '<F11>', dap.step_into)
vim.keymap.set('n', '<F12>', dap.step_out)
vim.keymap.set('n', '<leader>b', dap.toggle_breakpoint)
vim.keymap.set('n', '<leader>B', function()
  dap.set_breakpoint(vim.fn.input('Breakpoint condition: '))
end)
vim.keymap.set('n', '<leader>lp', function()
  dap.set_breakpoint(nil, nil, vim.fn.input('Log point message: '))
end)
vim.keymap.set('n', '<leader>dr', dap.repl.open)
vim.keymap.set('n', '<leader>dl', dap.run_last)
```

Performance Optimization

Startup Optimization

```
-- ~/.config/nvim/lua/performance.lua

-- Disable unused providers
vim.g.loaded_python_provider = 0
vim.g.loaded_ruby_provider = 0
vim.g.loaded_perl_provider = 0
vim.g.loaded_node_provider = 0

-- Lazy load plugins
vim.opt.runtimepath:prepend(vim.fn.stdpath("data") .. "/lazy/lazy.nvim")

-- Optimize for large files
vim.api.nvim_create_autocmd("BufReadPre", {
  callback = function()
    local file_size = vim.fn.getfsize(vim.fn.expand("%"))
    if file_size > 1024 * 1024 then -- 1MB
```

```
      vim.opt_local.syntax = "off"
      vim.opt_local.filetype = ""
      vim.opt_local.undolevels = -1
      vim.opt_local.swapfile = false
      vim.opt_local.loadplugins = false
    end
  end,
})
```

Migration from Vim

Compatibility Layer

```lua
-- ~/.config/nvim/lua/vim-compat.lua

-- Vim compatibility settings
vim.opt.compatible = false
vim.opt.encoding = 'utf-8'
vim.opt.fileencoding = 'utf-8'

-- Source existing vimrc if needed
local vimrc = vim.fn.expand("~/.vimrc")
if vim.fn.filereadable(vimrc) == 1 then
  vim.cmd("source " .. vimrc)
end

-- Convert common vim settings to lua
local function convert_vim_settings()
  -- Add your vim-to-lua conversions here
  if vim.g.loaded_vimrc then
    -- Convert specific vim settings
    vim.opt.number = vim.g.vim_number or false
    vim.opt.relativenumber = vim.g.vim_relativenumber or false
  end
end

convert_vim_settings()
```

Troubleshooting

Common Issues

```
-- ~/.config/nvim/lua/troubleshooting.lua

-- Health check
vim.keymap.set('n', '<leader>h', ':checkhealth<CR>')

-- Plugin debugging
vim.keymap.set('n', '<leader>pi', ':PackerInstall<CR>')
vim.keymap.set('n', '<leader>ps', ':PackerSync<CR>')
vim.keymap.set('n', '<leader>pc', ':PackerClean<CR>')

-- LSP debugging
vim.keymap.set('n', '<leader>li', ':LspInfo<CR>')
vim.keymap.set('n', '<leader>lr', ':LspRestart<CR>')

-- Log files
-- ~/.local/share/nvim/lsp.log
-- ~/.cache/nvim/packer.nvim.log
```

Neovim represents the future of modal editing, combining Vim's efficiency with modern development tools and practices. Its built-in LSP support, Lua scripting, and active plugin ecosystem make it an excellent choice for developers seeking a powerful, customizable editor.

Nano

Nano: The User-Friendly Terminal Editor

Nano is a simple, user-friendly text editor for Unix and Linux systems. Created as a free replacement for the Pico editor, nano provides an intuitive interface with on-screen help, making it ideal for beginners and quick edits.

Why Choose Nano?

- **Beginner-friendly**: Easy to learn with on-screen help
- **No modes**: Direct typing without modal complexity
- **Lightweight**: Minimal resource usage, perfect for Linux servers
- **Universal**: Available on all Linux distributions by default
- **Quick edits**: Perfect for Linux configuration files and system administration
- **Syntax highlighting**: Built-in support for many file types
- **System administration**: Ideal for editing Linux config files and scripts

Installation

Most Linux Distributions (Pre-installed)

```
# Check if nano is installed
which nano
nano --version

# If not installed:
# Ubuntu/Debian
sudo apt update
sudo apt install nano

# CentOS/RHEL/Fedora
sudo yum install nano
# or
sudo dnf install nano

# Arch Linux
sudo pacman -S nano

# Alpine Linux
```

```
sudo apk add nano
```

Additional Linux Distributions

```
# SUSE/openSUSE
sudo zypper install nano

# Gentoo
sudo emerge app-editors/nano

# Void Linux
sudo xbps-install nano
```

Basic Usage

Starting Nano

```
# Create new file
nano filename.txt

# Edit existing file
nano /path/to/file.txt

# Open file at specific line
nano +25 filename.txt

# Open file as read-only
nano -v filename.txt

# Open with line numbers
nano -l filename.txt

# Open with syntax highlighting disabled
nano -Y no filename.txt
```

Interface Overview

```
  GNU nano 6.2                    filename.txt                    Modified

This is the content of your file.
You can type directly here.
No modes to worry about!
```

```
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify
^X Exit        ^R Read File   ^\ Replace     ^U Paste Text  ^T To Spell
```

The bottom shows available commands where ^ means Ctrl key.

Essential Commands

File Operations

```
# Save file
Ctrl+O (Write Out)
# Then press Enter to confirm filename

# Save as different name
Ctrl+O
# Type new filename and press Enter

# Exit nano
Ctrl+X
# If unsaved changes, nano will ask to save

# Read/Insert another file
Ctrl+R
# Type filename to insert at cursor position

# New buffer (new file)
Alt+F (if multiple buffers enabled)
```

Navigation

```
# Basic movement
Arrow keys        # Move cursor
Ctrl+A           # Beginning of line
Ctrl+E           # End of line
Ctrl+Y           # Previous page
Ctrl+V           # Next page

# Advanced navigation
Ctrl+_           # Go to line number
Ctrl+W           # Search (Where Is)
Alt+W            # Search next occurrence
Ctrl+\           # Search and replace
```

```
# Word movement
Ctrl+Space       # Move forward one word
Alt+Space        # Move backward one word

# Beginning/End of file
Alt+\            # Go to beginning of file
Alt+/            # Go to end of file
```

Text Editing

```
# Cut and paste
Ctrl+K           # Cut current line
Ctrl+U           # Paste (uncut)
Alt+6            # Copy current line
Ctrl+6           # Mark text (start selection)
                 # Move cursor to select text
Ctrl+K           # Cut selected text

# Delete operations
Backspace        # Delete character before cursor
Delete           # Delete character at cursor
Alt+Backspace    # Delete word to the left
Ctrl+Delete      # Delete word to the right

# Undo/Redo
Alt+U            # Undo
Alt+E            # Redo
```

Search and Replace

```
# Search
Ctrl+W           # Open search prompt
# Type search term and press Enter
# Use Ctrl+W again to search for next occurrence

# Search and replace
Ctrl+\           # Open replace prompt
# Enter search term, press Enter
# Enter replacement term, press Enter
# Choose: Y (yes), N (no), A (all), Ctrl+C (cancel)

# Case-sensitive search
Alt+C            # Toggle case sensitivity during search
```

```
# Regular expression search
Alt+R            # Toggle regex mode during search
```

Configuration

Global Configuration

```
# System-wide configuration
/etc/nanorc

# User configuration
~/.nanorc
```

Sample .nanorc Configuration

```
# ~/.nanorc

## Use auto-indentation
set autoindent

## Use bold text instead of reverse video text
set boldtext

## Set the characters treated as closing brackets
set brackets "\"')>]}}"

## Do case-sensitive searches by default
set casesensitive

## Constantly display the cursor position in the statusbar
set constantshow

## Use cut-from-cursor-to-end-of-line by default
set cutfromcursor

## Set the line length for wrapping text and justifying paragraphs
set fill 72

## Remember the used search/replace strings for the next session
set historylog

## Display line numbers to the left of the text
set linenumbers
```

```
## Enable vim-style lock-files
set locking

## Don't display the helpful shortcut lists at the bottom of the screen
# set nohelp

## Don't automatically add a newline when a file doesn't end with one
set nonewlines

## Set operating directory (chroot of sorts)
# set operatingdir "/tmp"

## Remember the cursor position in each file for the next editing session
set positionlog

## Do extended regular expression searches by default
set regexp

## Make the Home key smarter
set smarthome

## Use smooth scrolling as the default
set smooth

## Use this spelling checker instead of the internal one
set speller "aspell -x -c"

## Allow nano to be suspended
set suspend

## Use this tab size instead of the default; it must be greater than 0
set tabsize 4

## Convert typed tabs to spaces
set tabstospaces

## Save automatically on exit; don't prompt
# set tempfile

## Disallow file modification; why would you want this in an rc file?
# set view

## The two single-column characters used to display the first characters
## of tabs and spaces
set whitespace "»·"
```

```
## Detect word boundaries more accurately by treating punctuation
## characters as parts of words
set wordbounds

## Color syntax highlighting
include "/usr/share/nano/*.nanorc"
```

Syntax Highlighting

```
# Enable syntax highlighting for specific file types
# Add to ~/.nanorc:

## Python
include "/usr/share/nano/python.nanorc"

## HTML
include "/usr/share/nano/html.nanorc"

## CSS
include "/usr/share/nano/css.nanorc"

## JavaScript
include "/usr/share/nano/javascript.nanorc"

## JSON
include "/usr/share/nano/json.nanorc"

## Markdown
include "/usr/share/nano/markdown.nanorc"

## Shell scripts
include "/usr/share/nano/sh.nanorc"

## Configuration files
include "/usr/share/nano/conf.nanorc"

## Or include all available syntax files
include "/usr/share/nano/*.nanorc"
```

Custom Syntax Highlighting

```
# ~/.config/nano/custom.nanorc

## Custom syntax for log files
syntax "log" "\.log$"
color red "ERROR.*"
color yellow "WARNING.*"
color green "INFO.*"
color blue "DEBUG.*"
color magenta "\[[0-9-]+ [0-9:]+\]"

## Custom syntax for configuration files
syntax "config" "\.(conf|config|cfg)$"
color green "^[[:space:]]*[^=]*="
color red "^[[:space:]]*#.*"
color yellow ""[^"]*""
color cyan "[0-9]+"
```

Advanced Features

Multiple Buffers

```
# Enable multiple buffers in ~/.nanorc
set multibuffer

# Commands for multiple buffers
Alt+F            # Switch to next buffer
Alt+B            # Switch to previous buffer
Ctrl+R           # Read file into new buffer
```

Spell Checking

```
# Install spell checker
sudo apt install aspell aspell-en

# Use spell check in nano
Ctrl+T           # Start spell check
# Navigate through misspelled words
# Choose replacement or skip
```

Backup Files

```
# Enable backup files in ~/.nanorc
set backup

# Set backup directory
set backupdir "/home/user/.nano/backups"

# Create backup directory
mkdir -p ~/.nano/backups
```

Mouse Support

```
# Enable mouse support in ~/.nanorc
set mouse

# Mouse functions:
# - Click to position cursor
# - Double-click to select word
# - Triple-click to select line
# - Scroll wheel to scroll text
```

Real-World Usage Scenarios

Scenario 1: Quick Configuration File Edit

```
# Edit SSH configuration
sudo nano /etc/ssh/sshd_config

# Navigate to specific setting
Ctrl+W
# Search for "PasswordAuthentication"

# Change the value
# Move cursor to "yes" and change to "no"

# Save and exit
Ctrl+O
Enter
Ctrl+X

# Restart SSH service
sudo systemctl restart sshd
```

Scenario 2: System Log Analysis

```
# View system log
sudo nano -v /var/log/syslog

# Search for errors
Ctrl+W
# Type "error" and press Enter

# Continue searching
Ctrl+W
Enter (to search for same term)

# Go to specific time
Ctrl+W
# Type timestamp like "2023-12-01 10:"

# Exit without changes
Ctrl+X
```

Scenario 3: Script Development

```
# Create new script
nano backup_script.sh

# Add shebang and content
#!/bin/bash
# Backup script
DATE=$(date +%Y%m%d)
tar -czf backup_$DATE.tar.gz /home/user/documents

# Enable line numbers for debugging
Alt+N

# Save and make executable
Ctrl+O
Enter
Ctrl+X
chmod +x backup_script.sh
```

Scenario 4: Crontab Editing

```
# Edit user crontab
crontab -e
# If nano is default editor, it opens in nano

# Add scheduled job
# 0 2 * * * /home/user/backup_script.sh

# Save crontab
Ctrl+O
Enter
Ctrl+X
```

Command Line Options

Useful Nano Options

```
# Open with line numbers
nano -l filename.txt

# Open in restricted mode (no file operations)
nano -R filename.txt

# Set tab width
nano -T 8 filename.txt

# Enable soft wrapping
nano -S filename.txt

# Disable wrapping
nano -w filename.txt

# Show cursor position
nano -c filename.txt

# Enable mouse support
nano -m filename.txt

# Ignore nanorc files
nano -I filename.txt

# Set operating directory
nano -o /tmp filename.txt
```

```
# Enable suspension
nano -z filename.txt
```

Environment Variables

```
# Set nano as default editor
export EDITOR=nano
export VISUAL=nano

# Add to ~/.bashrc or ~/.zshrc
echo 'export EDITOR=nano' >> ~/.bashrc
echo 'export VISUAL=nano' >> ~/.bashrc
```

Tips and Best Practices

Productivity Tips

1. **Learn the shortcuts**: Memorize Ctrl+O (save), Ctrl+X (exit), Ctrl+W (search)
2. **Use line numbers**: Enable with -l option or set linenumbers in .nanorc
3. **Configure syntax highlighting**: Include language-specific nanorc files
4. **Use search effectively**: Ctrl+W for quick navigation in large files
5. **Enable mouse support**: Makes selection and positioning easier

Common Shortcuts Summary

```
# File operations
Ctrl+O          # Save (Write Out)
Ctrl+X          # Exit
Ctrl+R          # Read/Insert file

# Navigation
Ctrl+A          # Beginning of line
Ctrl+E          # End of line
Ctrl+Y          # Previous page
Ctrl+V          # Next page
Ctrl+_          # Go to line number

# Editing
Ctrl+K          # Cut line
Ctrl+U          # Paste (Uncut)
Ctrl+6          # Mark text
Alt+U           # Undo
Alt+E           # Redo
```

```
# Search
Ctrl+W          # Search
Ctrl+\          # Replace
Alt+W           # Find next

# Help
Ctrl+G          # Get help
```

Customization for Different Use Cases

For Programming

```
# ~/.nanorc for programming
set autoindent
set linenumbers
set tabsize 4
set tabstospaces
set mouse
set smooth
set constantshow
include "/usr/share/nano/*.nanorc"
```

For System Administration

```
# ~/.nanorc for sysadmin
set backup
set backupdir "/home/user/.nano/backups"
set linenumbers
set constantshow
set mouse
set locking
include "/usr/share/nano/*.nanorc"
```

For Writing

```
# ~/.nanorc for writing
set fill 80
set justify
set smooth
set mouse
set speller "aspell -x -c"
set tempfile
```

Troubleshooting

Common Issues

1. **Nano not found**

   ```
   # Install nano
   sudo apt install nano
   ```

2. **No syntax highlighting**

   ```
   # Check if nanorc files exist
   ls /usr/share/nano/
   # Add to ~/.nanorc
   include "/usr/share/nano/*.nanorc"
   ```

3. **Can't save file (permission denied)**

   ```
   # Use sudo
   sudo nano /etc/hosts
   # Or change file permissions
   sudo chmod 644 filename.txt
   ```

4. **Accidental changes**

   ```
   # Exit without saving
   Ctrl+X
   # Choose 'N' when asked to save
   ```

5. **Lost in large file**

   ```
   # Go to specific line
   Ctrl+_
   # Enter line number
   ```

Recovery Options

```
# If nano crashes, look for backup files
ls ~/.nano/backups/
# or
ls /tmp/

# Recover from backup
nano ~/.nano/backups/filename.txt~
```

Nano excels as a straightforward, accessible text editor that doesn't require learning complex commands or modes. It's perfect for quick edits, system administration tasks, and users who prefer a simple, intuitive interface over the power and complexity of vim or emacs.

Emacs

Emacs: The Extensible Editor

GNU Emacs is one of the most powerful and extensible text editors available. Created by Richard Stallman in 1976, Emacs is more than just an editor—it's a complete computing environment that can be customized and extended using Emacs Lisp.

Why Choose Emacs?

- **Extreme extensibility**: Customize everything with Emacs Lisp
- **Integrated environment**: Email, calendar, file manager, shell, and more
- **Powerful editing**: Advanced text manipulation and programming features
- **Linux integration**: Deep integration with Linux development workflows
- **Active community**: Thousands of packages and configurations
- **Self-documenting**: Comprehensive built-in help system
- **Org-mode**: Powerful organization and note-taking system
- **System administration**: Excellent for managing Linux servers and configurations

Installation

Linux Distributions

```
# Ubuntu/Debian
sudo apt update
sudo apt install emacs

# For GUI version
sudo apt install emacs-gtk

# CentOS/RHEL/Fedora
sudo yum install emacs
# or
sudo dnf install emacs

# Arch Linux
sudo pacman -S emacs

# Alpine Linux
sudo apk add emacs
```

Additional Linux Distributions

```
# SUSE/openSUSE
sudo zypper install emacs

# Gentoo
sudo emerge app-editors/emacs

# Void Linux
sudo xbps-install emacs

# For GUI version on Linux
sudo apt install emacs-gtk    # Ubuntu/Debian with GTK
sudo pacman -S emacs          # Arch Linux (includes GUI)
```

Building from Source

```
# Install dependencies (Ubuntu/Debian)
sudo apt install build-essential texinfo libx11-dev libxpm-dev \
    libjpeg-dev libpng-dev libgif-dev libtiff-dev libgtk-3-dev \
    libncurses-dev libxpm-dev automake autoconf

# Download and build
wget https://ftp.gnu.org/gnu/emacs/emacs-29.1.tar.xz
tar -xf emacs-29.1.tar.xz
cd emacs-29.1
./configure --with-native-compilation
make -j$(nproc)
sudo make install
```

Basic Concepts

Key Notation

- `C-x` means Ctrl+x
- `M-x` means Alt+x (Meta key)
- `C-x C-f` means Ctrl+x followed by Ctrl+f
- `RET` means Enter key
- `SPC` means Space key

Buffers, Windows, and Frames

- **Buffer**: A file or content in memory

- **Window**: A view of a buffer (can have multiple windows)
- **Frame**: The entire Emacs window (can have multiple frames)

Major and Minor Modes

- **Major Mode**: Defines the primary editing behavior (e.g., Python mode, HTML mode)
- **Minor Mode**: Additional features that can be enabled/disabled (e.g., line numbers, auto-complete)

Essential Commands

Starting and Exiting

```
# Start Emacs
emacs

# Start in terminal mode
emacs -nw

# Open specific file
emacs filename.txt

# Exit Emacs
C-x C-c
```

File Operations

```
# Open file
C-x C-f
# Type filename and press RET

# Save file
C-x C-s

# Save as (write file)
C-x C-w

# Save all buffers
C-x s

# Insert file contents
C-x i
```

```
# Recent files
C-x C-r
```

Buffer Management

```
# Switch buffer
C-x b
# Type buffer name or use TAB completion

# List all buffers
C-x C-b

# Kill (close) buffer
C-x k

# Switch to next buffer
C-x →

# Switch to previous buffer
C-x ←
```

Window Management

```
# Split window horizontally
C-x 2

# Split window vertically
C-x 3

# Switch to other window
C-x o

# Delete current window
C-x 0

# Delete other windows
C-x 1

# Resize windows
C-x ^    # Make taller
C-x }    # Make wider
C-x {    # Make narrower
```

Navigation and Editing

Basic Movement

```
# Character movement
C-f      # Forward character
C-b      # Backward character
C-n      # Next line
C-p      # Previous line

# Word movement
M-f      # Forward word
M-b      # Backward word

# Line movement
C-a      # Beginning of line
C-e      # End of line

# Sentence movement
M-a      # Beginning of sentence
M-e      # End of sentence

# Paragraph movement
M-{      # Beginning of paragraph
M-}      # End of paragraph

# Page movement
C-v      # Page down
M-v      # Page up

# Buffer movement
M-<      # Beginning of buffer
M->      # End of buffer
```

Advanced Navigation

```
# Go to line
M-g g    # Go to line number
M-g M-g  # Alternative

# Go to character
M-g c    # Go to character position

# Jump to definition
M-.      # Find definition
```

```
M-,      # Return from definition

# Bookmarks
C-x r m  # Set bookmark
C-x r b  # Jump to bookmark
C-x r l  # List bookmarks
```

Text Editing

```
# Delete operations
C-d      # Delete character forward
DEL      # Delete character backward
M-d      # Delete word forward
M-DEL    # Delete word backward
C-k      # Kill to end of line
M-k      # Kill to end of sentence

# Kill and yank (cut and paste)
C-w      # Kill region
M-w      # Copy region
C-y      # Yank (paste)
M-y      # Cycle through kill ring

# Undo
C-/      # Undo
C-x u    # Undo
C-g      # Quit/cancel command
```

Selection (Regions)

```
# Set mark
C-SPC    # Set mark at cursor
C-x C-x  # Exchange point and mark

# Select all
C-x h    # Select entire buffer

# Rectangle selection
C-x r r  # Copy rectangle
C-x r k  # Kill rectangle
C-x r y  # Yank rectangle
C-x r t  # Replace rectangle with text
```

Search and Replace

Basic Search

```
# Incremental search forward
C-s
# Type search term, C-s for next occurrence

# Incremental search backward
C-r

# Word search
C-s C-w  # Search for word at cursor

# Regular expression search
C-M-s    # Forward regex search
C-M-r    # Backward regex search
```

Replace Operations

```
# Query replace
M-%
# Enter search term, then replacement
# y (yes), n (no), ! (all), q (quit)

# Regular expression replace
C-M-%

# Replace in region
# Select region first, then M-%
```

Advanced Search

```
# Occur (show all lines matching pattern)
M-s o

# Multi-occur (search in multiple buffers)
M-s M-o

# Grep in files
M-x grep
M-x rgrep    # Recursive grep
```

Configuration

Init File

Emacs configuration is stored in `~/.emacs.d/init.el` or `~/.emacs`:

```
;; ~/.emacs.d/init.el - Basic configuration

;; Package management
(require 'package)
(setq package-archives '(("melpa" . "https://melpa.org/packages/")
                         ("org" . "https://orgmode.org/elpa/")
                         ("elpa" . "https://elpa.gnu.org/packages/")))
(package-initialize)

;; Bootstrap use-package
(unless (package-installed-p 'use-package)
  (package-refresh-contents)
  (package-install 'use-package))

(eval-when-compile
  (require 'use-package))

;; Basic settings
(setq inhibit-startup-message t)        ; Disable startup screen
(setq visible-bell t)                   ; Visual bell instead of beep
(setq make-backup-files nil)            ; Disable backup files
(setq auto-save-default nil)            ; Disable auto-save

;; UI improvements
(tool-bar-mode -1)                      ; Disable toolbar
(menu-bar-mode -1)                      ; Disable menu bar
(scroll-bar-mode -1)                    ; Disable scroll bar
(global-display-line-numbers-mode 1)    ; Show line numbers
(column-number-mode 1)                  ; Show column numbers
(show-paren-mode 1)                     ; Highlight matching parentheses

;; Indentation
(setq-default indent-tabs-mode nil)     ; Use spaces instead of tabs
(setq-default tab-width 4)              ; Tab width
(setq indent-line-function 'insert-tab) ; Tab behavior

;; Font and theme
(set-face-attribute 'default nil :font "Fira Code-12")
(load-theme 'wombat t)

;; Key bindings
```

```
(global-set-key (kbd "C-x C-b") 'ibuffer)
(global-set-key (kbd "M-/") 'hippie-expand)
(global-set-key (kbd "C-c c") 'comment-or-uncomment-region)

;; Auto-completion
(use-package company
  :ensure t
  :init
  (global-company-mode 1))

;; Syntax checking
(use-package flycheck
  :ensure t
  :init
  (global-flycheck-mode))

;; Git integration
(use-package magit
  :ensure t
  :bind ("C-x g" . magit-status))

;; Project management
(use-package projectile
  :ensure t
  :init
  (projectile-mode +1)
  :bind-keymap
  ("C-c p" . projectile-command-map))

;; File tree
(use-package neotree
  :ensure t
  :bind ("F8" . neotree-toggle))

;; Better search
(use-package ivy
  :ensure t
  :init
  (ivy-mode 1)
  :config
  (setq ivy-use-virtual-buffers t)
  (setq enable-recursive-minibuffers t))

(use-package counsel
  :ensure t
  :bind (("M-x" . counsel-M-x)
         ("C-x C-f" . counsel-find-file)
```

```
         ("C-h f" . counsel-describe-function)
         ("C-h v" . counsel-describe-variable)))

(use-package swiper
  :ensure t
  :bind ("C-s" . swiper))
```

Package Management

```
;; Install packages manually
M-x package-install RET package-name RET

;; List packages
M-x list-packages

;; Update packages
M-x package-list-packages
# Press 'U' then 'x' to update all
```

Popular Packages

```
;; Essential packages configuration

;; Helm (alternative to ivy/counsel)
(use-package helm
  :ensure t
  :bind (("M-x" . helm-M-x)
         ("C-x C-f" . helm-find-files)
         ("C-x b" . helm-buffers-list))
  :config
  (helm-mode 1))

;; Which-key (show key bindings)
(use-package which-key
  :ensure t
  :config
  (which-key-mode))

;; Multiple cursors
(use-package multiple-cursors
  :ensure t
  :bind (("C-S-c C-S-c" . mc/edit-lines)
         ("C->" . mc/mark-next-like-this)
```

```
         ("C-<" . mc/mark-previous-like-this)))

;; Expand region
(use-package expand-region
  :ensure t
  :bind ("C-=" . er/expand-region))

;; Yasnippet (snippets)
(use-package yasnippet
  :ensure t
  :config
  (yas-global-mode 1))

;; Language Server Protocol
(use-package lsp-mode
  :ensure t
  :hook ((python-mode . lsp)
         (js-mode . lsp)
         (typescript-mode . lsp))
  :commands lsp)

(use-package lsp-ui
  :ensure t
  :commands lsp-ui-mode)

;; Company LSP integration
(use-package company-lsp
  :ensure t
  :commands company-lsp)
```

Programming with Emacs

Language-Specific Modes

```
;; Python development
(use-package python-mode
  :ensure t
  :mode "\\.py\\'"
  :config
  (setq python-indent-offset 4))

;; JavaScript/TypeScript
(use-package js2-mode
  :ensure t
```

```
  :mode "\\.js\\'"
  :config
  (setq js2-basic-offset 2))

(use-package typescript-mode
  :ensure t
  :mode "\\.ts\\'")

;; Web development
(use-package web-mode
  :ensure t
  :mode ("\\.html\\'" "\\.css\\'" "\\.jsx\\'")
  :config
  (setq web-mode-markup-indent-offset 2)
  (setq web-mode-css-indent-offset 2)
  (setq web-mode-code-indent-offset 2))

;; Markdown
(use-package markdown-mode
  :ensure t
  :mode "\\.md\\'")

;; YAML
(use-package yaml-mode
  :ensure t
  :mode "\\.ya?ml\\'")

;; JSON
(use-package json-mode
  :ensure t
  :mode "\\.json\\'")
```

Code Navigation

```
# Tags (requires etags or ctags)
M-.      # Find definition
M-,      # Return from definition
M-*      # Pop tag mark

# Imenu (function/class navigation)
M-x imenu

# Occur (find all occurrences)
M-s o
```

```
# Grep
M-x grep
M-x rgrep
```

Debugging

```
;; GDB integration
M-x gdb

;; Python debugging with pdb
M-x pdb

;; Edebug (Emacs Lisp debugger)
C-u C-M-x   ; Instrument function for debugging
```

Org Mode

Org-mode is one of Emacs' most powerful features for organization and note-taking:

Basic Org Syntax

```
* Top Level Heading
** Second Level Heading
*** Third Level Heading

- Unordered list item
- Another item
  - Sub-item

1. Ordered list item
2. Another ordered item

*bold* /italic/ _underlined_ =code= ~verbatim~

[[https://example.com][Link text]]

#+BEGIN_SRC python
def hello_world():
    print("Hello, World!")
#+END_SRC

| Name | Age | City |
```

```
|------+-----+------|
| John |  25 | NYC  |
| Jane |  30 | LA   |
```

Org Mode Commands

```
# Structure editing
TAB      # Cycle visibility
S-TAB    # Global visibility cycling
M-RET    # Insert new heading
M-←/→    # Promote/demote heading
M-↑/↓    # Move heading up/down

# TODO items
C-c C-t  # Cycle TODO states
C-c C-s  # Schedule item
C-c C-d  # Set deadline

# Tables
C-c |    # Create table
TAB      # Move to next field
S-TAB    # Move to previous field
C-c C-c  # Align table

# Links
C-c C-l  # Insert link
C-c C-o  # Open link

# Export
C-c C-e  # Export menu
```

Org Configuration

```
;; Org-mode configuration
(use-package org
  :ensure t
  :config
  (setq org-todo-keywords
        '((sequence "TODO" "IN-PROGRESS" "WAITING" "|" "DONE" "CANCELLED")))
  (setq org-agenda-files '("~/org/"))
  (setq org-default-notes-file "~/org/notes.org")
  :bind (("C-c l" . org-store-link)
         ("C-c a" . org-agenda)
         ("C-c c" . org-capture)))
```

```
;; Org capture templates
(setq org-capture-templates
      '(("t" "Todo" entry (file+headline "~/org/tasks.org" "Tasks")
         "* TODO %?\n  %i\n  %a")
        ("n" "Note" entry (file+headline "~/org/notes.org" "Notes")
         "* %?\nEntered on %U\n  %i\n  %a")))
```

Real-World Development Scenarios

Scenario 1: Python Development Setup

```
;; Python development configuration
(use-package python-mode
  :ensure t
  :config
  (setq python-indent-offset 4)
  (add-hook 'python-mode-hook
            (lambda ()
              (setq indent-tabs-mode nil)
              (setq tab-width 4)
              (setq python-indent-offset 4))))

;; Virtual environment support
(use-package pyvenv
  :ensure t
  :config
  (pyvenv-mode 1))

;; Python testing
(use-package python-pytest
  :ensure t
  :bind (:map python-mode-map
              ("C-c t" . python-pytest)))

;; Jupyter notebooks
(use-package ein
  :ensure t)
```

Scenario 2: Web Development Workflow

```
;; Web development setup
(use-package web-mode
  :ensure t
  :mode ("\\.html\\'" "\\.css\\'" "\\.js\\'" "\\.jsx\\'" "\\.tsx\\'")
  :config
  (setq web-mode-markup-indent-offset 2)
  (setq web-mode-css-indent-offset 2)
  (setq web-mode-code-indent-offset 2)
  (setq web-mode-enable-auto-pairing t)
  (setq web-mode-enable-css-colorization t))

;; Emmet for HTML expansion
(use-package emmet-mode
  :ensure t
  :hook (web-mode . emmet-mode))

;; Live browser reload
(use-package impatient-mode
  :ensure t)

;; REST client
(use-package restclient
  :ensure t
  :mode ("\\.http\\'" . restclient-mode))
```

Scenario 3: System Administration

```
;; System administration tools
(use-package tramp
  :config
  (setq tramp-default-method "ssh"))

;; Docker integration
(use-package docker
  :ensure t
  :bind ("C-c d" . docker))

(use-package dockerfile-mode
  :ensure t
  :mode "Dockerfile\\'")

;; Ansible
(use-package ansible
```

```
  :ensure t)

;; Log file viewing
(use-package logview
  :ensure t
  :mode ("\\.log\\'" . logview-mode))
```

Advanced Emacs Features

Macros

```
# Record macro
C-x (    # Start recording
# ... perform actions ...
C-x )    # Stop recording

# Execute macro
C-x e    # Execute once
C-u 10 C-x e  # Execute 10 times

# Name and save macro
M-x name-last-kbd-macro
M-x insert-kbd-macro
```

Registers

```
# Save to register
C-x r s a    # Save region to register 'a'
C-x r i a    # Insert from register 'a'

# Save position
C-x r SPC a  # Save position to register 'a'
C-x r j a    # Jump to register 'a'

# Save window configuration
C-x r w a    # Save window config to register 'a'
C-x r j a    # Restore window config from register 'a'
```

Dired (Directory Editor)

```
# Open dired
C-x d

# Dired commands
RET      # Open file/directory
q        # Quit dired
g        # Refresh
m        # Mark file
u        # Unmark file
d        # Mark for deletion
x        # Execute deletions
C        # Copy file
R        # Rename/move file
+        # Create directory
```

Shell Integration

```
# Shell commands
M-!      # Execute shell command
M-|      # Shell command on region
C-u M-!  # Insert command output

# Shell mode
M-x shell
M-x eshell   # Emacs shell
M-x term     # Terminal emulator
```

Customization and Themes

Custom Themes

```
;; Popular themes
(use-package doom-themes
  :ensure t
  :config
  (load-theme 'doom-one t))

(use-package solarized-theme
  :ensure t)

(use-package zenburn-theme
```

```
  :ensure t)

;; Theme switching function
(defun switch-theme (theme)
  "Disable current theme and load THEME."
  (interactive
   (list
    (intern (completing-read "Load custom theme: "
                             (mapcar 'symbol-name
                                     (custom-available-themes))))))
  (let ((enabled-themes custom-enabled-themes))
    (mapc #'disable-theme custom-enabled-themes)
    (load-theme theme t)))
```

Custom Functions

```
;; Useful custom functions
(defun duplicate-line ()
  "Duplicate current line."
  (interactive)
  (move-beginning-of-line 1)
  (kill-line)
  (yank)
  (open-line 1)
  (next-line 1)
  (yank))

(global-set-key (kbd "C-c d") 'duplicate-line)

(defun comment-or-uncomment-region-or-line ()
  "Comments or uncomments the region or the current line if there's no active region."
  (interactive)
  (let (beg end)
    (if (region-active-p)
        (setq beg (region-beginning) end (region-end))
      (setq beg (line-beginning-position) end (line-end-position)))
    (comment-or-uncomment-region beg end)))

(global-set-key (kbd "C-/") 'comment-or-uncomment-region-or-line)
```

Performance Optimization

Startup Optimization

```
;; Startup optimization
(setq gc-cons-threshold (* 50 1000 1000))  ; Increase GC threshold
(setq read-process-output-max (* 1024 1024)) ; Increase read buffer

;; Restore GC threshold after startup
(add-hook 'emacs-startup-hook
          (lambda ()
            (setq gc-cons-threshold (* 2 1000 1000))))

;; Lazy loading
(use-package package-name
  :ensure t
  :defer t  ; Lazy load
  :hook (mode . package-name))
```

Large File Handling

```
;; Large file handling
(defun my-find-file-check-make-large-file-read-only-hook ()
  "If a file is over a given size, make the buffer read only."
  (when (> (buffer-size) (* 1024 1024))
    (setq buffer-read-only t)
    (buffer-disable-undo)
    (fundamental-mode)))

(add-hook 'find-file-hook 'my-find-file-check-make-large-file-read-only-hook)
```

Troubleshooting

Common Issues

```
# Debug init file
emacs --debug-init

# Start with minimal config
emacs -Q

# Profile startup time
M-x emacs-init-time
```

```
# Check package issues
M-x package-list-packages
# Look for packages with issues

# Reset package system
# Delete ~/.emacs.d/elpa/ and restart
```

Recovery

```
# Auto-save recovery
M-x recover-file

# Session recovery
M-x desktop-save-mode  ; Enable session saving
M-x desktop-read       ; Restore session
```

Emacs is incredibly powerful and customizable, making it suitable for everything from simple text editing to complex development workflows. While it has a steep learning curve, the investment pays off with a highly personalized and efficient editing environment.

Lazyvim

LazyVim: Modern Neovim Configuration

LazyVim is a popular, pre-configured Neovim setup that provides a modern IDE-like experience out of the box. Built on top of lazy.nvim plugin manager, it offers sensible defaults, beautiful UI, and powerful features for developers.

What is LazyVim?

LazyVim is a Neovim configuration framework that includes: - **Pre-configured plugins**: Carefully selected and configured plugins - **Lazy loading**: Fast startup times with lazy.nvim - **Modern UI**: Beautiful interface with proper theming - **LSP integration**: Built-in Language Server Protocol support - **Extensible**: Easy to customize and extend - **Well-documented**: Comprehensive documentation and examples

Why Choose LazyVim?

- **Zero configuration**: Works great out of the box
- **Fast startup**: Optimized for performance
- **Modern features**: Latest Neovim capabilities
- **Active development**: Regular updates and improvements
- **Great defaults**: Sensible keybindings and settings
- **Easy customization**: Simple configuration structure

Installation

Prerequisites

```
# Neovim 0.9.0 or higher
nvim --version

# Install Neovim if needed
# Ubuntu/Debian
sudo apt install neovim

# Arch Linux
sudo pacman -S neovim
```

```
# SUSE/openSUSE
sudo zypper install neovim

# Gentoo
sudo emerge app-editors/neovim
```

Required Dependencies

```
# Git (required)
git --version

# Node.js (for LSP servers)
node --version
npm --version

# Python (for some plugins)
python3 --version
pip3 --version

# Ripgrep (for telescope)
rg --version

# fd (for telescope)
fd --version

# Install missing dependencies
# Ubuntu/Debian
sudo apt install git nodejs npm python3 python3-pip ripgrep fd-find

# Arch Linux
sudo pacman -S git nodejs npm python python-pip ripgrep fd

# SUSE/openSUSE
sudo zypper install git nodejs npm python3 python3-pip ripgrep fd

# Gentoo
sudo emerge dev-vcs/git net-libs/nodejs dev-lang/python sys-apps/ripgrep sys-apps/fd
```

Installation Steps

```
# Backup existing Neovim configuration
mv ~/.config/nvim ~/.config/nvim.bak
mv ~/.local/share/nvim ~/.local/share/nvim.bak
```

```
mv ~/.local/state/nvim ~/.local/state/nvim.bak
mv ~/.cache/nvim ~/.cache/nvim.bak

# Clone LazyVim starter template
git clone https://github.com/LazyVim/starter ~/.config/nvim

# Remove .git folder to make it your own
rm -rf ~/.config/nvim/.git

# Start Neovim (plugins will install automatically)
nvim
```

Initial Setup and Configuration

First Launch

When you first start LazyVim:

1. **Plugin Installation**: Plugins install automatically
2. **Mason Setup**: Language servers install automatically
3. **Treesitter**: Syntax parsers download automatically
4. **Health Check**: Run `:checkhealth` to verify setup

Basic Configuration Structure

```
~/.config/nvim/
  init.lua                  # Entry point
  lua/
      config/
          autocmds.lua     # Auto commands
          keymaps.lua      # Key mappings
          lazy.lua         # Plugin manager setup
          options.lua      # Neovim options
      plugins/
           example.lua      # Example plugin config
           ...              # Your plugin configurations
  stylua.toml               # Lua formatter config
```

Basic Options Configuration

```
-- ~/.config/nvim/lua/config/options.lua

-- LazyVim auto format
vim.g.autoformat = true
```

```lua
-- LazyVim picker to use
-- Can be one of: telescope, fzf
vim.g.lazyvim_picker = "telescope"

-- LazyVim root dir detection
-- Each entry can be:
-- * the name of a detector function like `lsp` or `cwd`
-- * a pattern or array of patterns like `.git` or `lua`.
-- * a function with signature `function(buf) -> string|string[]`
vim.g.root_spec = { "lsp", { ".git", "lua" }, "cwd" }

-- LazyVim automatically configures lazygit:
--  * theme, based on the active colorscheme.
--  * editorPreset to nvim-remote
--  * enables nerd font icons
-- Set to false to disable.
vim.g.lazygit_config = true

-- Options are automatically loaded before lazy.nvim startup
-- Default options that are always set: https://github.com/LazyVim/LazyVim/blob/main/lua/laz
-- Add any additional options here

-- Line numbers
vim.opt.relativenumber = true -- Relative line numbers
vim.opt.number = true -- Print line number

-- Tabs / Indentation
vim.opt.tabstop = 2 -- 2 spaces for tabs (prettier default)
vim.opt.shiftwidth = 2 -- 2 spaces for indent width
vim.opt.expandtab = true -- expand tab to spaces
vim.opt.autoindent = true -- copy indent from current line when starting new one

-- Line wrapping
vim.opt.wrap = false -- disable line wrapping

-- Search settings
vim.opt.ignorecase = true -- ignore case when searching
vim.opt.smartcase = true -- if you include mixed case in your search, assumes you want case-

-- Cursor line
vim.opt.cursorline = true -- highlight the current cursor line

-- Appearance
vim.opt.termguicolors = true
vim.opt.background = "dark" -- colorschemes that can be light or dark will be made dark
vim.opt.signcolumn = "yes" -- show sign column so that text doesn't shift
```

```
-- Backspace
vim.opt.backspace = "indent,eol,start" -- allow backspace on indent, end of line or insert m

-- Clipboard
vim.opt.clipboard:append("unnamedplus") -- use system clipboard as default register

-- Split windows
vim.opt.splitright = true -- split vertical window to the right
vim.opt.splitbelow = true -- split horizontal window to the bottom

-- Turn off swapfile
vim.opt.swapfile = false
```

Custom Keymaps

```
-- ~/.config/nvim/lua/config/keymaps.lua

-- Keymaps are automatically loaded on the VeryLazy event
-- Default keymaps that are always set: https://github.com/LazyVim/LazyVim/blob/main/lua/laz
-- Add any additional keymaps here

local keymap = vim.keymap -- for conciseness

-- General keymaps
keymap.set("i", "jk", "<ESC>", { desc = "Exit insert mode with jk" })

keymap.set("n", "<leader>nh", ":nohl<CR>", { desc = "Clear search highlights" })

-- increment/decrement numbers
keymap.set("n", "<leader>+", "<C-a>", { desc = "Increment number" }) -- increment
keymap.set("n", "<leader>-", "<C-x>", { desc = "Decrement number" }) -- decrement

-- window management
keymap.set("n", "<leader>sv", "<C-w>v", { desc = "Split window vertically" }) -- split windo
keymap.set("n", "<leader>sh", "<C-w>s", { desc = "Split window horizontally" }) -- split win
keymap.set("n", "<leader>se", "<C-w>=", { desc = "Make splits equal size" }) -- make split w
keymap.set("n", "<leader>sx", "<cmd>close<CR>", { desc = "Close current split" }) -- close c

keymap.set("n", "<leader>to", "<cmd>tabnew<CR>", { desc = "Open new tab" }) -- open new tab
keymap.set("n", "<leader>tx", "<cmd>tabclose<CR>", { desc = "Close current tab" }) -- close
keymap.set("n", "<leader>tn", "<cmd>tabn<CR>", { desc = "Go to next tab" }) --  go to next t
keymap.set("n", "<leader>tp", "<cmd>tabp<CR>", { desc = "Go to previous tab" }) --  go to pr
keymap.set("n", "<leader>tf", "<cmd>tabnew %<CR>", { desc = "Open current buffer in new tab"

-- Move text up and down
```

```lua
keymap.set("v", "J", ":m '>+1<CR>gv=gv", { desc = "Move text down" })
keymap.set("v", "K", ":m '<-2<CR>gv=gv", { desc = "Move text up" })

-- Stay in indent mode
keymap.set("v", "<", "<gv", { desc = "Indent left" })
keymap.set("v", ">", ">gv", { desc = "Indent right" })

-- Keep last yanked when pasting
keymap.set("v", "p", '"_dP', { desc = "Paste without yanking" })
```

Auto Commands

```lua
-- ~/.config/nvim/lua/config/autocmds.lua

-- Autocmds are automatically loaded on the VeryLazy event
-- Default autocmds that are always set: https://github.com/LazyVim/LazyVim/blob/main/lua/la
-- Add any additional autocmds here

local function augroup(name)
  return vim.api.nvim_create_augroup("lazyvim_" .. name, { clear = true })
end

-- Highlight on yank
vim.api.nvim_create_autocmd("TextYankPost", {
  group = augroup("highlight_yank"),
  callback = function()
    vim.highlight.on_yank()
  end,
})

-- resize splits if window got resized
vim.api.nvim_create_autocmd({ "VimResized" }, {
  group = augroup("resize_splits"),
  callback = function()
    local current_tab = vim.fn.tabpagenr()
    vim.cmd("tabdo wincmd =")
    vim.cmd("tabnext " .. current_tab)
  end,
})

-- go to last loc when opening a buffer
vim.api.nvim_create_autocmd("BufReadPost", {
  group = augroup("last_loc"),
  callback = function(event)
    local exclude = { "gitcommit" }
```

```
    local buf = event.buf
    if vim.tbl_contains(exclude, vim.bo[buf].filetype) or vim.b[buf].lazyvim_last_loc then
      return
    end
    vim.b[buf].lazyvim_last_loc = true
    local mark = vim.api.nvim_buf_get_mark(buf, '"')
    local lcount = vim.api.nvim_buf_line_count(buf)
    if mark[1] > 0 and mark[1] <= lcount then
      pcall(vim.api.nvim_win_set_cursor, 0, mark)
    end
  end,
})

-- close some filetypes with <q>
vim.api.nvim_create_autocmd("FileType", {
  group = augroup("close_with_q"),
  pattern = {
    "PlenaryTestPopup",
    "help",
    "lspinfo",
    "man",
    "notify",
    "qf",
    "query",
    "spectre_panel",
    "startuptime",
    "tsplayground",
    "neotest-output",
    "checkhealth",
    "neotest-summary",
    "neotest-output-panel",
  },
  callback = function(event)
    vim.bo[event.buf].buflisted = false
    vim.keymap.set("n", "q", "<cmd>close<cr>", { buffer = event.buf, silent = true })
  end,
})

-- wrap and check for spell in text filetypes
vim.api.nvim_create_autocmd("FileType", {
  group = augroup("wrap_spell"),
  pattern = { "gitcommit", "markdown" },
  callback = function()
    vim.opt_local.wrap = true
    vim.opt_local.spell = true
  end,
})
```

```lua
-- Auto create dir when saving a file, in case some intermediate directory does not exist
vim.api.nvim_create_autocmd({ "BufWritePre" }, {
  group = augroup("auto_create_dir"),
  callback = function(event)
    if event.match:match("^%w%w+://") then
      return
    end
    local file = vim.loop.fs_realpath(event.match) or event.match
    vim.fn.mkdir(vim.fn.fnamemodify(file, ":p:h"), "p")
  end,
})
```

Plugin Configuration

Adding Custom Plugins

```lua
-- ~/.config/nvim/lua/plugins/example.lua

return {
  -- Add your custom plugins here
  {
    "folke/flash.nvim",
    event = "VeryLazy",
    opts = {},
    keys = {
      {
        "s",
        mode = { "n", "x", "o" },
        function()
          require("flash").jump()
        end,
        desc = "Flash",
      },
      {
        "S",
        mode = { "n", "o", "x" },
        function()
          require("flash").treesitter()
        end,
        desc = "Flash Treesitter",
      },
    },
  },
```

```
  -- Disable default plugins
  { "folke/flash.nvim", enabled = false },

  -- Override plugin configuration
  {
    "nvim-telescope/telescope.nvim",
    opts = {
      defaults = {
        layout_strategy = "horizontal",
        layout_config = { prompt_position = "top" },
        sorting_strategy = "ascending",
        winblend = 0,
      },
    },
  },
}
```

Language-Specific Configurations

```
-- ~/.config/nvim/lua/plugins/languages.lua

return {
  -- Python
  {
    "nvim-treesitter/nvim-treesitter",
    opts = function(_, opts)
      if type(opts.ensure_installed) == "table" then
        vim.list_extend(opts.ensure_installed, { "python" })
      end
    end,
  },
  {
    "neovim/nvim-lspconfig",
    opts = {
      servers = {
        pyright = {},
        ruff_lsp = {},
      },
    },
  },

  -- JavaScript/TypeScript
  {
    "nvim-treesitter/nvim-treesitter",
    opts = function(_, opts)
```

```
    if type(opts.ensure_installed) == "table" then
      vim.list_extend(opts.ensure_installed, { "javascript", "typescript", "tsx" })
    end
  end,
},
{
  "neovim/nvim-lspconfig",
  opts = {
    servers = {
      tsserver = {},
      eslint = {},
    },
  },
},

-- Go
{
  "nvim-treesitter/nvim-treesitter",
  opts = function(_, opts)
    if type(opts.ensure_installed) == "table" then
      vim.list_extend(opts.ensure_installed, { "go", "gomod", "gowork", "gosum" })
    end
  end,
},
{
  "neovim/nvim-lspconfig",
  opts = {
    servers = {
      gopls = {},
    },
  },
},

-- Rust
{
  "nvim-treesitter/nvim-treesitter",
  opts = function(_, opts)
    if type(opts.ensure_installed) == "table" then
      vim.list_extend(opts.ensure_installed, { "rust", "toml" })
    end
  end,
},
{
  "neovim/nvim-lspconfig",
  opts = {
    servers = {
```

```
        rust_analyzer = {},
      },
    },
  },
}
```

Essential LazyVim Features

File Explorer (Neo-tree)

```
# Default keymaps
<leader>fe    # Open file explorer
<leader>fE    # Open file explorer (root dir)
<leader>e     # Toggle file explorer
<leader>E     # Toggle file explorer (root dir)

# Neo-tree navigation
j/k           # Move up/down
h/l           # Collapse/expand or enter
<CR>          # Open file
o             # Open file
s             # Open in split
v             # Open in vsplit
t             # Open in tab
a             # Add file/directory
d             # Delete
r             # Rename
c             # Copy
x             # Cut
p             # Paste
q             # Close
```

Fuzzy Finding (Telescope)

```
# File finding
<leader>ff    # Find files
<leader>fr    # Recent files
<leader>fg    # Find files (git)
<leader>fF    # Find files (cwd)

# Text searching
<leader>sg    # Grep search
<leader>sw    # Search word under cursor
```

```
<leader>sG    # Grep search (cwd)
<leader>ss    # Search symbols
<leader>sS    # Search symbols (workspace)

# Buffer management
<leader>fb    # Find buffers
<leader>,     # Switch buffers

# Git integration
<leader>gc    # Git commits
<leader>gs    # Git status

# Help and commands
<leader>sh    # Help pages
<leader>sk    # Key maps
<leader>sc    # Commands
<leader>sC    # Command history
```

LSP Features

```
# Navigation
gd            # Go to definition
gr            # Go to references
gI            # Go to implementation
gy            # Go to type definition
gD            # Go to declaration

# Information
K             # Hover information
gK            # Signature help
<c-k>         # Signature help (insert mode)

# Code actions
<leader>ca    # Code actions
<leader>cA    # Source actions
<leader>cr    # Rename
<leader>cf    # Format
<leader>cF    # Format (injected langs)

# Diagnostics
<leader>cd    # Line diagnostics
<leader>cD    # Workspace diagnostics
]d            # Next diagnostic
[d            # Previous diagnostic
]e            # Next error
```

```
[e            # Previous error
]w            # Next warning
[w            # Previous warning
```

Git Integration (Lazygit)

```
# Lazygit
<leader>gg    # Open Lazygit
<leader>gG    # Open Lazygit (root dir)

# Git signs (gitsigns)
]h            # Next hunk
[h            # Previous hunk
<leader>ghs   # Stage hunk
<leader>ghr   # Reset hunk
<leader>ghS   # Stage buffer
<leader>ghu   # Undo stage hunk
<leader>ghR   # Reset buffer
<leader>ghp   # Preview hunk
<leader>ghb   # Blame line
<leader>ghl   # Blame line (full)
<leader>ghd   # Diff this
<leader>ghD   # Diff this ~
```

Terminal Integration

```
# Terminal
<leader>ft    # Terminal (root dir)
<leader>fT    # Terminal (cwd)
<c-/>         # Terminal (root dir)
<c-_>         # Terminal (root dir) (which-key)

# Terminal navigation (in terminal mode)
<esc><esc>    # Enter normal mode
<C-h>         # Go to left window
<C-j>         # Go to lower window
<C-k>         # Go to upper window
<C-l>         # Go to right window
```

Real-World Development Setups

Python Development

```lua
-- ~/.config/nvim/lua/plugins/python.lua

return {
  -- Python-specific plugins
  {
    "linux-cultist/venv-selector.nvim",
    dependencies = { "neovim/nvim-lspconfig", "nvim-telescope/telescope.nvim", "mfussenegger
    opts = {
      -- Your options go here
      name = "venv",
      auto_refresh = false,
    },
    event = "VeryLazy", -- Optional: needed only if you want to type `:VenvSelect` without a
    keys = {
      -- Keymap to open VenvSelector to pick a venv.
      { "<leader>vs", "<cmd>VenvSelect<cr>" },
      -- Keymap to retrieve the venv from a cache (the one previously used for the same proj
      { "<leader>vc", "<cmd>VenvSelectCached<cr>" },
    },
  },

  -- Python debugging
  {
    "mfussenegger/nvim-dap-python",
    ft = "python",
    dependencies = {
      "mfussenegger/nvim-dap",
      "rcarriga/nvim-dap-ui",
    },
    config = function(_, opts)
      local path = "~/.local/share/nvim/mason/packages/debugpy/venv/bin/python"
      require("dap-python").setup(path)
    end,
  },

  -- Python testing
  {
    "nvim-neotest/neotest",
    optional = true,
    dependencies = {
      "nvim-neotest/neotest-python",
    },
```

```
  opts = {
    adapters = {
      ["neotest-python"] = {
        -- Here you can specify the settings for the adapter, i.e.
        runner = "pytest",
        python = ".venv/bin/python",
      },
    },
  },
},

-- Enhanced Python syntax
{
  "nvim-treesitter/nvim-treesitter",
  opts = function(_, opts)
    if type(opts.ensure_installed) == "table" then
      vim.list_extend(opts.ensure_installed, { "ninja", "python", "rst", "toml" })
    end
  end,
},

-- Python LSP configuration
{
  "neovim/nvim-lspconfig",
  opts = {
    servers = {
      pyright = {
        settings = {
          python = {
            analysis = {
              typeCheckingMode = "basic",
              autoSearchPaths = true,
              useLibraryCodeForTypes = true,
            },
          },
        },
      },
      ruff_lsp = {
        init_options = {
          settings = {
            args = {},
          },
        },
      },
    },
  },
},
```

```
}
```

Web Development

```
-- ~/.config/nvim/lua/plugins/web.lua

return {
  -- TypeScript/JavaScript
  {
    "neovim/nvim-lspconfig",
    opts = {
      servers = {
        tsserver = {
          settings = {
            typescript = {
              inlayHints = {
                includeInlayParameterNameHints = "literal",
                includeInlayParameterNameHintsWhenArgumentMatchesName = false,
                includeInlayFunctionParameterTypeHints = true,
                includeInlayVariableTypeHints = false,
                includeInlayPropertyDeclarationTypeHints = true,
                includeInlayFunctionLikeReturnTypeHints = true,
                includeInlayEnumMemberValueHints = true,
              },
            },
            javascript = {
              inlayHints = {
                includeInlayParameterNameHints = "all",
                includeInlayParameterNameHintsWhenArgumentMatchesName = false,
                includeInlayFunctionParameterTypeHints = true,
                includeInlayVariableTypeHints = true,
                includeInlayPropertyDeclarationTypeHints = true,
                includeInlayFunctionLikeReturnTypeHints = true,
                includeInlayEnumMemberValueHints = true,
              },
            },
          },
        },
        eslint = {},
        html = {},
        cssls = {},
        tailwindcss = {},
      },
    },
  },
```

```lua
-- Emmet for HTML/CSS
{
  "mattn/emmet-vim",
  ft = { "html", "css", "javascript", "typescript", "javascriptreact", "typescriptreact" }
},

-- Auto close tags
{
  "windwp/nvim-ts-autotag",
  ft = { "html", "javascript", "typescript", "javascriptreact", "typescriptreact", "svelte
  config = function()
    require("nvim-ts-autotag").setup()
  end,
},

-- Package.json management
{
  "vuki656/package-info.nvim",
  dependencies = "MunifTanjim/nui.nvim",
  ft = "json",
  config = function()
    require("package-info").setup()
  end,
},

-- REST client
{
  "rest-nvim/rest.nvim",
  dependencies = { "nvim-lua/plenary.nvim" },
  ft = "http",
  config = function()
    require("rest-nvim").setup({
      result_split_horizontal = false,
      result_split_in_place = false,
      skip_ssl_verification = false,
      encode_url = true,
      highlight = {
        enabled = true,
        timeout = 150,
      },
      result = {
        show_url = true,
        show_curl_command = false,
        show_http_info = true,
        show_headers = true,
        formatters = {
```

```
            json = "jq",
            html = function(body)
              return vim.fn.system({ "tidy", "-i", "-q", "-" }, body)
            end,
          },
        },
      })
    end,
  },
}
```

Go Development

```
-- ~/.config/nvim/lua/plugins/go.lua

return {
  -- Go plugin
  {
    "ray-x/go.nvim",
    dependencies = { -- optional packages
      "ray-x/guihua.lua",
      "neovim/nvim-lspconfig",
      "nvim-treesitter/nvim-treesitter",
    },
    config = function()
      require("go").setup()
    end,
    event = { "CmdlineEnter" },
    ft = { "go", "gomod" },
    build = ':lua require("go.install").update_all_sync()', -- if you need to install/update
  },

  -- Go testing
  {
    "nvim-neotest/neotest",
    optional = true,
    dependencies = {
      "nvim-neotest/neotest-go",
    },
    opts = {
      adapters = {
        ["neotest-go"] = {
          -- Here you can specify the settings for the adapter, i.e.
          args = { "-count=1", "-timeout=60s" },
        },
```

```
    },
  },
},

-- Go LSP configuration
{
  "neovim/nvim-lspconfig",
  opts = {
    servers = {
      gopls = {
        settings = {
          gopls = {
            gofumpt = true,
            codelenses = {
              gc_details = false,
              generate = true,
              regenerate_cgo = true,
              run_govulncheck = true,
              test = true,
              tidy = true,
              upgrade_dependency = true,
              vendor = true,
            },
            hints = {
              assignVariableTypes = true,
              compositeLiteralFields = true,
              compositeLiteralTypes = true,
              constantValues = true,
              functionTypeParameters = true,
              parameterNames = true,
              rangeVariableTypes = true,
            },
            analyses = {
              fieldalignment = true,
              nilness = true,
              unusedparams = true,
              unusedwrite = true,
              useany = true,
            },
            usePlaceholders = true,
            completeUnimported = true,
            staticcheck = true,
            directoryFilters = { "-.git", "-.vscode", "-.idea", "-.vscode-test", "-node_mo
            semanticTokens = true,
          },
        },
      },
```

```
      },
    },
  },

  -- Go debugging
  {
    "leoluz/nvim-dap-go",
    ft = "go",
    dependencies = "mfussenegger/nvim-dap",
    config = function(_, opts)
      require("dap-go").setup(opts)
    end,
  },
}
```

Customization Examples

Custom Colorscheme

```
-- ~/.config/nvim/lua/plugins/colorscheme.lua

return {
  -- Catppuccin theme
  {
    "catppuccin/nvim",
    name = "catppuccin",
    priority = 1000,
    opts = {
      flavour = "mocha", -- latte, frappe, macchiato, mocha
      background = { -- :h background
        light = "latte",
        dark = "mocha",
      },
      transparent_background = false,
      show_end_of_buffer = false,
      term_colors = false,
      dim_inactive = {
        enabled = false,
        shade = "dark",
        percentage = 0.15,
      },
      no_italic = false,
      no_bold = false,
      no_underline = false,
      styles = {
```

```
        comments = { "italic" },
        conditionals = { "italic" },
        loops = {},
        functions = {},
        keywords = {},
        strings = {},
        variables = {},
        numbers = {},
        booleans = {},
        properties = {},
        types = {},
        operators = {},
      },
      color_overrides = {},
      custom_highlights = {},
      integrations = {
        cmp = true,
        gitsigns = true,
        nvimtree = true,
        telescope = true,
        notify = false,
        mini = false,
      },
    },
  },

  -- Configure LazyVim to load catppuccin
  {
    "LazyVim/LazyVim",
    opts = {
      colorscheme = "catppuccin",
    },
  },
}
```

Custom Dashboard

```
-- ~/.config/nvim/lua/plugins/dashboard.lua

return {
  {
    "nvimdev/dashboard-nvim",
    event = "VimEnter",
    opts = function()
      local logo = [[
```

```
                                        Z
                                     Z
                                  z
                               z


]]

logo = string.rep("\n", 8) .. logo .. "\n\n"

local opts = {
  theme = "doom",
  hide = {
    statusline = false,
  },
  config = {
    header = vim.split(logo, "\n"),
    center = {
      { action = "Telescope find_files", desc = " Find file", icon = " ", key = "f" },
      { action = "ene | startinsert", desc = " New file", icon = " ", key = "n" },
      { action = "Telescope oldfiles", desc = " Recent files", icon = " ", key = "r" }
      { action = "Telescope live_grep", desc = " Find text", icon = " ", key = "g" },
      { action = [[lua require("lazyvim.util").telescope.config_files()()]], desc = "
      { action = 'lua require("persistence").load()', desc = " Restore Session", icon
      { action = "LazyExtras", desc = " Lazy Extras", icon = " ", key = "x" },
      { action = "Lazy", desc = " Lazy", icon = "  ", key = "l" },
      { action = "qa", desc = " Quit", icon = " ", key = "q" },
    },
    footer = function()
      local stats = require("lazy").stats()
      local ms = (math.floor(stats.startuptime * 100 + 0.5) / 100)
      return { "  Neovim loaded " .. stats.loaded .. "/" .. stats.count .. " plugins in
    end,
  },
}

for _, button in ipairs(opts.config.center) do
  button.desc = button.desc .. string.rep(" ", 43 - #button.desc)
  button.key_format = "  %s"
end

-- close Lazy and re-open when the dashboard is ready
if vim.o.filetype == "lazy" then
  vim.cmd.close()
  vim.api.nvim_create_autocmd("User", {
    pattern = "DashboardLoaded",
    callback = function()
```

```
            require("lazy").show()
          end,
        })
      end

      return opts
    end,
  },
}
```

Performance Optimization

Startup Optimization

```
-- ~/.config/nvim/lua/config/lazy.lua

-- Bootstrap lazy.nvim
local lazypath = vim.fn.stdpath("data") .. "/lazy/lazy.nvim"
if not vim.loop.fs_stat(lazypath) then
  vim.fn.system({
    "git",
    "clone",
    "--filter=blob:none",
    "https://github.com/folke/lazy.nvim.git",
    "--branch=stable",
    lazypath,
  })
end
vim.opt.rtp:prepend(lazypath)

require("lazy").setup({
  spec = {
    -- add LazyVim and import its plugins
    { "LazyVim/LazyVim", import = "lazyvim.plugins" },
    -- import any extras modules here
    { import = "lazyvim.plugins.extras.lang.typescript" },
    { import = "lazyvim.plugins.extras.lang.json" },
    { import = "lazyvim.plugins.extras.ui.mini-animate" },
    -- import/override with your plugins
    { import = "plugins" },
  },
  defaults = {
    lazy = false, -- should plugins be lazy-loaded?
    version = false, -- always use the latest git commit
```

```
  },
  install = { colorscheme = { "tokyonight", "habamax" } },
  checker = { enabled = true }, -- automatically check for plugin updates
  performance = {
    rtp = {
      -- disable some rtp plugins
      disabled_plugins = {
        "gzip",
        "matchit",
        "matchparen",
        "netrwPlugin",
        "tarPlugin",
        "tohtml",
        "tutor",
        "zipPlugin",
      },
    },
  },
})
```

Memory Optimization

```
-- ~/.config/nvim/lua/plugins/performance.lua

return {
  -- Optimize treesitter
  {
    "nvim-treesitter/nvim-treesitter",
    opts = {
      highlight = { enable = true },
      indent = { enable = true },
      ensure_installed = {
        "bash",
        "c",
        "html",
        "javascript",
        "json",
        "lua",
        "luadoc",
        "luap",
        "markdown",
        "markdown_inline",
        "python",
        "query",
        "regex",
```

```
        "tsx",
        "typescript",
        "vim",
        "vimdoc",
        "yaml",
      },
      incremental_selection = {
        enable = true,
        keymaps = {
          init_selection = "<C-space>",
          node_incremental = "<C-space>",
          scope_incremental = false,
          node_decremental = "<bs>",
        },
      },
    },
  },

  -- Optimize LSP
  {
    "neovim/nvim-lspconfig",
    opts = {
      diagnostics = {
        underline = true,
        update_in_insert = false,
        virtual_text = {
          spacing = 4,
          source = "if_many",
          prefix = " ",
        },
        severity_sort = true,
      },
      inlay_hints = {
        enabled = true,
      },
      capabilities = {},
      format = {
        formatting_options = nil,
        timeout_ms = nil,
      },
      servers = {},
      setup = {},
    },
  },
}
```

Troubleshooting

Common Issues

```
# Check LazyVim health
:checkhealth

# Check specific components
:checkhealth lazy
:checkhealth mason
:checkhealth telescope
:checkhealth treesitter

# Plugin management
:Lazy                  # Open Lazy plugin manager
:Lazy sync            # Update all plugins
:Lazy clean           # Remove unused plugins
:Lazy profile         # Profile startup time

# LSP troubleshooting
:LspInfo              # Show LSP information
:LspRestart           # Restart LSP servers
:Mason                # Open Mason (LSP installer)

# Clear cache
:lua vim.fn.delete(vim.fn.stdpath("cache"), "rf")
```

Reset Configuration

```
# Backup current config
mv ~/.config/nvim ~/.config/nvim.backup
mv ~/.local/share/nvim ~/.local/share/nvim.backup
mv ~/.local/state/nvim ~/.local/state/nvim.backup
mv ~/.cache/nvim ~/.cache/nvim.backup

# Fresh install
git clone https://github.com/LazyVim/starter ~/.config/nvim
rm -rf ~/.config/nvim/.git
```

Performance Issues

```
-- Profile startup time
-- Add to ~/.config/nvim/lua/config/options.lua
vim.g.loaded_python3_provider = 0
vim.g.loaded_ruby_provider = 0
vim.g.loaded_perl_provider = 0
vim.g.loaded_node_provider = 0

-- Disable unused features
vim.g.loaded_gzip = 1
vim.g.loaded_zip = 1
vim.g.loaded_zipPlugin = 1
vim.g.loaded_tar = 1
vim.g.loaded_tarPlugin = 1
vim.g.loaded_getscript = 1
vim.g.loaded_getscriptPlugin = 1
vim.g.loaded_vimball = 1
vim.g.loaded_vimballPlugin = 1
vim.g.loaded_2html_plugin = 1
vim.g.loaded_logiPat = 1
vim.g.loaded_rrhelper = 1
vim.g.loaded_netrw = 1
vim.g.loaded_netrwPlugin = 1
vim.g.loaded_netrwSettings = 1
vim.g.loaded_netrwFileHandlers = 1
```

LazyVim provides an excellent foundation for a modern Neovim setup, combining the power of Neovim with sensible defaults and a beautiful interface. It's perfect for developers who want a powerful IDE-like experience without the complexity of building a configuration from scratch.

Helix

Helix: A Modern Modal Editor

Helix is a modern, terminal-based text editor written in Rust. It takes inspiration from Kakoune and Vim, featuring multiple selections, built-in LSP support, and a focus on simplicity and performance.

What is Helix?

Helix is designed with modern editing principles: - **Selection-first editing**: Select text first, then act on it - **Multiple selections**: Edit multiple locations simultaneously - **Built-in LSP**: Language Server Protocol support out of the box - **Tree-sitter**: Advanced syntax highlighting and text objects - **No configuration required**: Works great with zero configuration - **Fast and lightweight**: Written in Rust for performance

Why Choose Helix?

- **Modern design**: Built from the ground up with modern editing concepts
- **Zero configuration**: Sensible defaults that work immediately
- **Multiple selections**: Powerful multi-cursor editing
- **Built-in features**: LSP, tree-sitter, and fuzzy finder included
- **Consistent keybindings**: Logical and memorable key combinations
- **Linux-optimized**: Designed with Linux development workflows in mind
- **Active development**: Regular updates and improvements
- **Rust performance**: Fast and memory-efficient on Linux systems

Installation

Package Managers

```
# Ubuntu/Debian (from source)
sudo apt update
sudo apt install git rustc cargo
git clone https://github.com/helix-editor/helix
cd helix
cargo install --path helix-term --locked

# Arch Linux
```

```
sudo pacman -S helix

# Fedora
sudo dnf install helix

# SUSE/openSUSE
sudo zypper install helix

# Gentoo
sudo emerge app-editors/helix

# Void Linux
sudo xbps-install helix

# Nix
nix-env -iA nixpkgs.helix

# Cargo (from source)
cargo install helix-term --locked
```

Building from Source

```
# Clone repository
git clone https://github.com/helix-editor/helix
cd helix

# Build and install
cargo install --path helix-term --locked

# Install runtime files
mkdir -p ~/.config/helix
cp -r runtime ~/.config/helix/

# Or use system-wide installation
sudo mkdir -p /usr/local/lib/helix
sudo cp -r runtime /usr/local/lib/helix/
```

Post-Installation Setup

```
# Install language servers (optional but recommended)
# JavaScript/TypeScript
npm install -g typescript-language-server typescript
```

```
# Python
pip install python-lsp-server

# Rust (if not already installed)
rustup component add rust-analyzer

# Go
go install golang.org/x/tools/gopls@latest

# Check health and see what's available
hx --health
```

Basic Concepts

Selection-First Editing

Unlike Vim's action-first approach, Helix uses selection-first:

```
# Vim way: action + motion
dw    # Delete word (action first)

# Helix way: selection + action
w     # Select word
d     # Delete selection
```

Multiple Selections

Helix excels at multiple selections:

```
# Select all occurrences of current selection
%     # Select all matches

# Add selection
C     # Duplicate cursor/selection
Alt-C # Remove primary selection

# Split selections
S     # Split selection on regex
Alt-s # Split selection on lines
```

Modes

Helix has several modes: - **Normal mode**: Default mode for navigation and commands - **Insert mode**: For typing text - **Select mode**: For making selections - **Command mode**: For ex-style

Essential Commands

Starting Helix

```
# Open file
hx filename.txt

# Open multiple files
hx file1.txt file2.txt

# Open directory (file picker)
hx .

# Open with specific line/column
hx filename.txt:25:10

# Health check
hx --health
```

Basic Navigation

```
# Character movement
h     # Left
j     # Down
k     # Up
l     # Right

# Word movement
w     # Next word start
b     # Previous word start
e     # Next word end

# Line movement
0     # Line start
^     # First non-whitespace
$     # Line end
g h   # Line start (alternative)
g l   # Line end (alternative)

# Document movement
g g   # Document start
g e   # Document end
```

```
Ctrl+u # Page up
Ctrl+d # Page down

# Paragraph movement
[      # Previous paragraph
]      # Next paragraph
```

Text Selection

```
# Basic selection
v      # Enter select mode
V      # Select line
Ctrl+v # Select block (not implemented yet)

# Word/object selection
w      # Select word
W      # Select WORD (whitespace-separated)
p      # Select paragraph
f<char> # Select to character
t<char> # Select until character

# Extend selection
;      # Flip selection direction
Alt+; # Ensure forward selection direction
%      # Select matching bracket
m      # Select matching pair (quotes, brackets, etc.)

# Multiple selections
C      # Duplicate cursor
Alt+C # Remove primary cursor
s      # Select regex matches within selection
S      # Split selection on regex
Alt-s # Split selection on newlines
&      # Align selections
_      # Trim whitespace from selections
```

Editing Commands

```
# Insert modes
i      # Insert before selection
a      # Insert after selection
I      # Insert at line start
A      # Insert at line end
o      # Insert new line below
```

```
O     # Insert new line above

# Delete/Change
d     # Delete selection
c     # Change selection (delete and enter insert mode)
Alt+d # Delete line
Alt+c # Change line

# Copy/Paste
y     # Yank (copy) selection
p     # Paste after selection
P     # Paste before selection
Alt+p # Paste every yanked selection
R     # Replace selection with yanked text

# Undo/Redo
u     # Undo
U     # Redo
Alt+u # Earlier in time
Alt+U # Later in time
```

Search and Replace

```
# Search
/     # Search forward
?     # Search backward
n     # Next search result
N     # Previous search result
*     # Search word under cursor

# Replace
r<char> # Replace character
~     # Switch case
`     # Set case to lowercase
Alt+` # Set case to uppercase

# Global replace (in command mode)
:s/old/new/g        # Replace in current selection
:%s/old/new/g       # Replace in entire file
:s/old/new/gc       # Replace with confirmation
```

File and Buffer Management

File Operations

```
# File commands (in command mode)
:o filename   # Open file
:w            # Write file
:w filename   # Write as filename
:q            # Quit
:q!           # Quit without saving
:wq           # Write and quit

# Buffer navigation
Space b       # Buffer picker
gn            # Next buffer
gp            # Previous buffer
Space x       # Close buffer
```

File Picker and Fuzzy Finding

```
# File operations
Space f       # File picker
Space F       # File picker (from cwd)
Space b       # Buffer picker
Space s       # Symbol picker (LSP)
Space S       # Workspace symbol picker
Space g       # Global search (grep)
Space /       # Global search in current file
Space k       # Show documentation
Space r       # Rename symbol (LSP)
Space a       # Code action (LSP)
```

Configuration

Configuration File

Helix uses TOML for configuration:

```
# ~/.config/helix/config.toml

theme = "onedark"

[editor]
```

```
line-number = "relative"
mouse = true
middle-click-paste = true
scroll-lines = 3
shell = ["sh", "-c"]
text-width = 80
completion-trigger-len = 2
auto-completion = true
auto-format = true
auto-save = false
idle-timeout = 400
preview-completion-insert = true
completion-replace = false
auto-info = true
true-color = false
rulers = [80, 120]
bufferline = "always"
color-modes = false

[editor.statusline]
left = ["mode", "spinner"]
center = ["file-name"]
right = ["diagnostics", "selections", "position", "file-encoding", "file-line-ending", "file
separator = " "
mode.normal = "NORMAL"
mode.insert = "INSERT"
mode.select = "SELECT"

[editor.lsp]
display-messages = true
auto-signature-help = true
display-inlay-hints = true
display-signature-help-docs = true
snippets = true
goto-reference-include-declaration = true

[editor.cursor-shape]
insert = "bar"
normal = "block"
select = "underline"

[editor.file-picker]
hidden = true
follow-symlinks = true
deduplicate-links = true
parents = true
ignore = true
```

```
git-ignore = true
git-global = true
git-exclude = true
max-depth = 25

[editor.auto-pairs]
'(' = ')'
'{' = '}'
'[' = ']'
'"' = '"'
"'" = "'"
"`" = "`"
'<' = '>'

[editor.search]
smart-case = true
wrap-around = true

[editor.whitespace]
render = "all"
characters = { space = "·", nbsp = " ", tab = "→", newline = " ", tabpad = "·" }

[editor.indent-guides]
render = true
character = " "
skip-levels = 1

[editor.gutters]
layout = ["diff", "diagnostics", "line-numbers", "spacer"]

[editor.soft-wrap]
enable = false
max-wrap = 25
max-indent-retain = 0
wrap-indicator = ""

[keys.normal]
# Custom keybindings
C-s = ":w"
C-o = "file_picker"
C-p = "file_picker"
esc = ["collapse_selection", "keep_primary_selection"]

[keys.insert]
# Insert mode keybindings
C-s = ["normal_mode", ":w", "insert_mode"]
j = { k = "normal_mode" } # jk to exit insert mode
```

```
[keys.select]
# Select mode keybindings
esc = ["collapse_selection", "keep_primary_selection", "normal_mode"]
```

Language Configuration

```
# ~/.config/helix/languages.toml

# Override language server configurations
[[language]]
name = "python"
language-server = { command = "pylsp" }
auto-format = true

[language.config.pylsp.plugins]
autopep8 = { enabled = false }
flake8 = { enabled = true }
mccabe = { enabled = false }
pycodestyle = { enabled = false }
pyflakes = { enabled = false }
pylint = { enabled = true }
yapf = { enabled = true }

[[language]]
name = "javascript"
language-server = { command = "typescript-language-server", args = ["--stdio"] }
auto-format = true

[[language]]
name = "typescript"
language-server = { command = "typescript-language-server", args = ["--stdio"] }
auto-format = true

[[language]]
name = "rust"
language-server = { command = "rust-analyzer" }
auto-format = true

[[language]]
name = "go"
language-server = { command = "gopls" }
auto-format = true

[[language]]
name = "html"
```

```
language-server = { command = "vscode-html-language-server", args = ["--stdio"] }
auto-format = true

[[language]]
name = "css"
language-server = { command = "vscode-css-language-server", args = ["--stdio"] }
auto-format = true

[[language]]
name = "json"
language-server = { command = "vscode-json-language-server", args = ["--stdio"] }
auto-format = true

# Custom language definitions
[[language]]
name = "log"
scope = "text.log"
file-types = ["log"]
comment-token = "#"
indent = { tab-width = 2, unit = "  " }

[[grammar]]
name = "log"
source = { git = "https://github.com/Tudyx/tree-sitter-log", rev = "main" }
```

Themes

Helix comes with many built-in themes:

```
# List available themes
:theme

# Change theme temporarily
:theme dracula

# Popular themes
:theme onedark
:theme gruvbox
:theme catppuccin_mocha
:theme tokyo-night
:theme solarized_dark
:theme nord
```

Custom Theme

```
# ~/.config/helix/themes/mytheme.toml

"ui.background" = { bg = "#1e1e1e" }
"ui.text" = "#d4d4d4"
"ui.text.focus" = "#ffffff"
"ui.selection" = { bg = "#264f78" }
"ui.cursor" = { fg = "#1e1e1e", bg = "#d4d4d4" }
"ui.cursor.match" = { bg = "#3a3d41" }
"ui.linenr" = "#858585"
"ui.linenr.selected" = "#c6c6c6"
"ui.statusline" = { fg = "#d4d4d4", bg = "#007acc" }
"ui.statusline.inactive" = { fg = "#d4d4d4", bg = "#3c3c3c" }
"ui.popup" = { bg = "#252526" }
"ui.window" = "#3c3c3c"
"ui.help" = { bg = "#252526" }

"error" = "#f44747"
"warning" = "#ff8c00"
"info" = "#0080ff"
"hint" = "#969696"

"diagnostic.error" = { underline = { color = "#f44747", style = "curl" } }
"diagnostic.warning" = { underline = { color = "#ff8c00", style = "curl" } }
"diagnostic.info" = { underline = { color = "#0080ff", style = "curl" } }
"diagnostic.hint" = { underline = { color = "#969696", style = "curl" } }

"syntax.comment" = "#6a9955"
"syntax.keyword" = "#569cd6"
"syntax.string" = "#ce9178"
"syntax.number" = "#b5cea8"
"syntax.boolean" = "#569cd6"
"syntax.type" = "#4ec9b0"
"syntax.function" = "#dcdcaa"
"syntax.variable" = "#9cdcfe"
"syntax.constant" = "#4fc1ff"
"syntax.operator" = "#d4d4d4"
"syntax.punctuation" = "#d4d4d4"
```

Real-World Usage Scenarios

Scenario 1: Python Development

```
# Open Python project
hx main.py

# Navigate to function definition
gd    # Go to definition (LSP)
gr    # Go to references (LSP)

# Multiple cursor editing
# Select function name
w     # Select word
%     # Select all occurrences
c     # Change all occurrences
new_function_name<Esc>

# Code formatting
:format   # Format current file
# Or enable auto-format in config

# Symbol search
Space s   # Search symbols in file
Space S   # Search symbols in workspace

# Diagnostics
]d        # Next diagnostic
[d        # Previous diagnostic
Space k   # Show hover information
Space a   # Code actions
```

Scenario 2: Web Development

```
# Open HTML file
hx index.html

# Select HTML tag
mt    # Select matching tag pair
c     # Change tag
div<Esc>

# Multiple selections for attributes
# Select class attribute value
f"    # Find quote
```

```
mi    # Select inside quotes
%     # Select all similar
c     # Change all
new-class<Esc>

# Navigate between files
Space f   # File picker
# Type filename and Enter

# Global search and replace
Space g   # Global search
# Type search term
# Navigate results with j/k
# Press Enter to go to location
```

Scenario 3: Configuration File Editing

```
# Open config file
hx ~/.config/helix/config.toml

# Navigate to specific section
/editor   # Search for "editor"
n         # Next occurrence

# Select and modify values
f=        # Find equals sign
l         # Move right
w         # Select value
c         # Change value
new_value<Esc>

# Comment/uncomment lines
Alt+;     # Toggle line comment
# Or select multiple lines and comment
V         # Select line
jjj       # Extend selection
Alt+;     # Comment selected lines
```

Scenario 4: Log File Analysis

```
# Open log file
hx /var/log/application.log

# Search for errors
```

```
/ERROR    # Search for ERROR
n         # Next occurrence
N         # Previous occurrence

# Select all error lines
*         # Search word under cursor (ERROR)
%         # Select all matches
y         # Yank all error lines

# Open new buffer for analysis
:new
p         # Paste error lines

# Split selections by timestamp
s^\d{4}-\d{2}-\d{2}  # Split on date pattern
```

Advanced Features

Multiple Selections Mastery

```
# Complex selection scenarios
# Select all function calls
/\w+\(   # Search for function calls
%        # Select all matches

# Select inside parentheses for all matches
mi       # Select inside matching pairs

# Add numbers to each selection
C        # Duplicate cursor to each selection
I        # Insert at beginning
1. <Esc> # Add numbering (manual)

# Or use regex replacement
:s/^/1. /  # Add numbering with regex
```

Text Objects

```
# Built-in text objects
mw    # Select word
mp    # Select paragraph
m(    # Select inside parentheses
m)    # Select around parentheses
```

```
m[    # Select inside brackets
m]    # Select around brackets
m{    # Select inside braces
m}    # Select around braces
m"    # Select inside quotes
m'    # Select inside single quotes
m`    # Select inside backticks
mt    # Select inside HTML/XML tags
mf    # Select function
mc    # Select class
ma    # Select argument
```

Tree-sitter Navigation

```
# Navigate syntax tree
Alt+p # Parent node
Alt+c # Child node
Alt+n # Next sibling
Alt+o # Previous sibling

# Select syntax nodes
mf    # Select function
mc    # Select class
ma    # Select argument/parameter
```

LSP Features

```
# Code navigation
gd    # Go to definition
gy    # Go to type definition
gr    # Go to references
gi    # Go to implementation

# Code information
Space k   # Hover information
Space K   # Signature help

# Code modification
Space r   # Rename symbol
Space a   # Code actions
:format   # Format document

# Diagnostics
]d        # Next diagnostic
```

```
[d        # Previous diagnostic
]D        # Next error
[D        # Previous error
Space d   # Show diagnostics
```

Customization and Scripting

Custom Commands

```
# ~/.config/helix/config.toml

[keys.normal]
# Custom command sequences
C-g = [":new", ":insert-output echo 'Hello World'"]
C-t = [":sh date"]

# Macro-like sequences
F1 = ["select_all", "yank", ":new", "paste_after"]
F2 = ["goto_line_start", "insert_mode", "// ", "normal_mode"]
```

Integration with External Tools

```
# Pipe selection to external command
|sort     # Sort selected lines
|uniq     # Remove duplicate lines
|wc -l    # Count lines
|jq       # Format JSON
|xmllint --format -  # Format XML

# Insert command output
:insert-output date
:insert-output ls -la
:insert-output git log --oneline -10
```

Shell Integration

```
# Set Helix as default editor
export EDITOR=hx
export VISUAL=hx

# Git integration
```

```
git config --global core.editor hx

# Shell aliases
alias h='hx'
alias hx.='hx .'
alias hxc='hx ~/.config/helix/config.toml'
```

Performance and Optimization

Large File Handling

Helix handles large files efficiently:

```
# Open large files
hx large_file.log

# Navigate efficiently
gg    # Go to start
ge    # Go to end
Ctrl+u # Page up
Ctrl+d # Page down

# Search in large files
/pattern  # Incremental search
n         # Next match
N         # Previous match
```

Memory Usage

```
# Check memory usage
:debug-info

# Optimize for memory
# Disable unnecessary features in config
[editor]
auto-completion = false
auto-info = false
```

Troubleshooting

Common Issues

```
# Check health
hx --health

# Language server issues
:lsp-workspace-command
:lsp-restart

# Theme issues
:theme default
:reload-config

# Key binding conflicts
# Check config.toml for conflicting bindings
```

Debug Mode

```
# Start with debug logging
RUST_LOG=debug hx

# Check logs
tail -f ~/.cache/helix/helix.log
```

Reset Configuration

```
# Backup current config
mv ~/.config/helix ~/.config/helix.backup

# Start fresh
hx  # Will create default config
```

Migration from Other Editors

From Vim/Neovim

Key differences to remember: - Selection-first editing - Different keybindings for some operations - Built-in LSP and tree-sitter - No plugin system (yet)

From VSCode

Helix provides many VSCode-like features: - Built-in LSP support - Fuzzy file finding - Multiple selections - Integrated terminal (external)

From Kakoune

Helix is inspired by Kakoune: - Similar selection-first philosophy - Multiple selections - Different keybindings and features

Helix offers a modern, efficient editing experience with powerful features built-in. Its selection-first approach and multiple selections make it particularly effective for complex text manipulation tasks, while the zero-configuration philosophy means you can be productive immediately.

Micro

Micro: A Modern Terminal Editor

Micro is a modern, intuitive terminal-based text editor that aims to be easy to use and easy to remember. It provides a familiar interface similar to GUI editors while running in the terminal, making it perfect for users transitioning from graphical editors.

What is Micro?

Micro is designed with simplicity and usability in mind: - **Intuitive keybindings**: Familiar Ctrl+S, Ctrl+C, Ctrl+V shortcuts - **Mouse support**: Click to position cursor, select text, scroll - **Syntax highlighting**: Built-in support for 100+ languages - **Plugin system**: Extensible with Lua plugins - **Multiple cursors**: Edit multiple locations simultaneously - **Linux-optimized**: Excellent performance on Linux systems - **No configuration required**: Works great out of the box

Why Choose Micro?

- **Easy to learn**: Familiar keybindings from GUI editors
- **Modern features**: Multiple cursors, syntax highlighting, plugins
- **Lightweight**: Single binary with no dependencies, perfect for Linux servers
- **Customizable**: Extensive configuration options
- **Active development**: Regular updates and improvements
- **Terminal-based**: Works over SSH and in minimal Linux environments
- **Unicode support**: Full Unicode and emoji support
- **System administration**: Great for Linux configuration file editing

Installation

Package Managers

```
# Ubuntu/Debian
sudo apt update
sudo apt install micro

# Fedora
sudo dnf install micro

# Arch Linux
```

```
sudo pacman -S micro

# SUSE/openSUSE
sudo zypper install micro

# Gentoo
sudo emerge app-editors/micro

# Void Linux
sudo xbps-install micro

# Snap (Universal Linux)
sudo snap install micro --classic

# Flatpak
flatpak install flathub io.github.micro-editor.micro
```

Direct Download

```
# Download latest release
curl https://getmic.ro | bash

# Or manually download
wget https://github.com/zyedidia/micro/releases/download/v2.0.11/micro-2.0.11-linux64.tar.gz
tar -xzf micro-2.0.11-linux64.tar.gz
sudo mv micro-2.0.11/micro /usr/local/bin/

# Make executable
chmod +x /usr/local/bin/micro
```

From Source

```
# Install Go if not already installed
# Then build micro
git clone https://github.com/zyedidia/micro
cd micro
make build
sudo mv micro /usr/local/bin/
```

Basic Usage

Starting Micro

```
# Open new file
micro

# Open existing file
micro filename.txt

# Open multiple files
micro file1.txt file2.txt

# Open directory (file browser)
micro .

# Open with specific settings
micro -colorscheme monokai filename.txt
micro -syntax python script.py
```

Interface Overview

```
  filename.txt
This is the content of your file.
You can type directly here.
Mouse clicks work for positioning cursor.









Ctrl+Q Quit  Ctrl+S Save  Ctrl+O Open  Ctrl+F Find  Ctrl+Z Undo  Ctrl+Y Redo
```

Essential Keybindings

File Operations

```
# File management
Ctrl+O          # Open file
Ctrl+S          # Save file
Ctrl+W          # Close current buffer
Ctrl+Q          # Quit micro
Ctrl+E          # Command mode
Ctrl+G          # Open help

# Buffer navigation
Ctrl+Tab        # Next buffer
Ctrl+Shift+Tab  # Previous buffer
Alt+,           # Previous buffer
Alt+.           # Next buffer
```

Navigation

```
# Basic movement
Arrow keys      # Move cursor
Ctrl+Left/Right # Move by word
Ctrl+Up/Down    # Move by paragraph
Home/End        # Beginning/end of line
Ctrl+Home/End   # Beginning/end of file
Page Up/Down    # Page navigation

# Advanced navigation
Ctrl+G          # Go to line
Ctrl+F          # Find
Ctrl+N          # Find next
Ctrl+P          # Find previous
```

Text Editing

```
# Basic editing
Ctrl+Z          # Undo
Ctrl+Y          # Redo
Ctrl+C          # Copy
Ctrl+X          # Cut
Ctrl+V          # Paste
Ctrl+A          # Select all
```

```
Ctrl+D          # Duplicate line
Ctrl+K          # Delete line

# Advanced editing
Ctrl+L          # Select line
Ctrl+Shift+Up   # Move line up
Ctrl+Shift+Down # Move line down
Alt+Shift+Left  # Select word left
Alt+Shift+Right # Select word right
```

Multiple Cursors

```
# Multiple cursor operations
Ctrl+MouseClick # Add cursor at click position
Alt+N           # Create cursor at next word occurrence
Alt+P           # Create cursor at previous word occurrence
Alt+C           # Create cursor at all word occurrences
Alt+M           # Remove all cursors except primary
Escape          # Remove all extra cursors
```

Search and Replace

```
# Search operations
Ctrl+F          # Find
Ctrl+N          # Find next
Ctrl+P          # Find previous
Ctrl+R          # Find and replace

# In find/replace dialog
Tab             # Switch between find and replace fields
Enter           # Execute search/replace
Escape          # Cancel dialog
```

Configuration

Settings File

Micro stores settings in `~/.config/micro/settings.json`:

```
{
    "autoclose": true,
    "autoindent": true,
```

```
    "autosave": 0,
    "autosu": false,
    "backup": true,
    "backupdir": "",
    "basename": false,
    "colorcolumn": 0,
    "colorscheme": "default",
    "cursorline": true,
    "diffgutter": false,
    "divchars": "|-",
    "divreverse": true,
    "encoding": "utf-8",
    "eofnewline": true,
    "fastdirty": false,
    "fileformat": "unix",
    "filetype": "unknown",
    "hlsearch": false,
    "ignorecase": false,
    "indentchar": " ",
    "infobar": true,
    "keepautoindent": false,
    "keymenu": false,
    "matchbrace": true,
    "matchbraceleft": false,
    "mouse": true,
    "parsecursor": false,
    "paste": false,
    "pluginchannels": [
        "https://raw.githubusercontent.com/micro-editor/plugin-channel/master/channel.json"
    ],
    "pluginrepos": [],
    "readonly": false,
    "rmtrailingws": false,
    "ruler": true,
    "savecursor": false,
    "savehistory": true,
    "saveundo": false,
    "scrollbar": false,
    "scrollmargin": 3,
    "scrollspeed": 2,
    "smartpaste": true,
    "softwrap": false,
    "splitbottom": true,
    "splitright": true,
    "statusformatl": "$(filename) $(modified)($(line),$(col)) $(status.paste)| ft:$(opt:file
    "statusformatr": "$(bind:ToggleKeyMenu): bindings, $(bind:ToggleHelp): help",
    "statusline": true,
```

```
    "sucmd": "sudo",
    "syntax": true,
    "tabmovement": false,
    "tabsize": 4,
    "tabstospaces": false,
    "termtitle": false,
    "useprimary": true,
    "wordwrap": false,
    "xterm": false
}
```

Common Configuration Changes

```
# Set tab size to 2
micro -option tabsize 2

# Enable line numbers
micro -option ruler true

# Set color scheme
micro -option colorscheme monokai

# Enable soft wrapping
micro -option softwrap true

# Auto-save every 10 seconds
micro -option autosave 10

# Remove trailing whitespace on save
micro -option rmtrailingws true

# Make changes permanent
# These settings are automatically saved to settings.json
```

Keybinding Customization

Create `~/.config/micro/bindings.json`:

```
{
    "Alt+/": "lua:comment.comment",
    "Ctrl+/": "lua:comment.comment",
    "Ctrl+Shift+D": "DuplicateLine",
    "Ctrl+Shift+K": "DeleteLine",
    "Ctrl+Shift+Up": "MoveLinesUp",
```

```
    "Ctrl+Shift+Down": "MoveLinesDown",
    "Ctrl+J": "JumpLine",
    "Ctrl+B": "ToggleBookmark",
    "F3": "FindNext",
    "Shift+F3": "FindPrevious",
    "F4": "Quit",
    "Ctrl+T": "NewTab",
    "Ctrl+Shift+T": "ReopenLastClosedTab"
}
```

Color Schemes

Micro comes with many built-in color schemes:

```
# List available color schemes
micro -option colorscheme help

# Popular color schemes
micro -option colorscheme monokai
micro -option colorscheme solarized
micro -option colorscheme dracula
micro -option colorscheme gruvbox
micro -option colorscheme atom-dark
micro -option colorscheme github
micro -option colorscheme zenburn
```

Custom Color Scheme

Create `~/.config/micro/colorschemes/mytheme.micro`:

```
color-link default "#d4d4d4,#1e1e1e"
color-link comment "#6a9955"
color-link identifier "#9cdcfe"
color-link constant "#4fc1ff"
color-link statement "#569cd6"
color-link symbol "#d4d4d4"
color-link preproc "#c586c0"
color-link type "#4ec9b0"
color-link special "#dcdcaa"
color-link underlined "#d4d4d4"
color-link error "#f44747"
color-link todo "#ffcc00"
color-link hlsearch "#264f78"
color-link statusline "#d4d4d4,#007acc"
color-link tabbar "#d4d4d4,#2d2d30"
```

```
color-link indent-char "#3c3c3c"
color-link line-number "#858585"
color-link current-line-number "#c6c6c6"
color-link diff-added "#28a745"
color-link diff-modified "#ffd33d"
color-link diff-deleted "#d73a49"
color-link gutter-error "#f44747"
color-link gutter-warning "#ff8c00"
color-link cursor-line "#2a2d2e"
color-link color-column "#3c3c3c"
```

Plugin System

Plugin Management

```
# List installed plugins
micro -plugin list

# Install plugin
micro -plugin install pluginname

# Remove plugin
micro -plugin remove pluginname

# Update plugins
micro -plugin update

# Search for plugins
micro -plugin search keyword

# Plugin information
micro -plugin info pluginname
```

Popular Plugins

```
# Essential plugins
micro -plugin install filemanager    # File browser
micro -plugin install linter         # Code linting
micro -plugin install comment        # Toggle comments
micro -plugin install jump           # Quick navigation
micro -plugin install manipulator    # Text manipulation
micro -plugin install quoter         # Quote manipulation
micro -plugin install snippets       # Code snippets
micro -plugin install autoclose      # Auto-close brackets
```

```
micro -plugin install bookmark       # Bookmarks
micro -plugin install diff           # Diff viewer

# Language-specific plugins
micro -plugin install go             # Go language support
micro -plugin install python         # Python support
micro -plugin install html           # HTML support
micro -plugin install css            # CSS support
micro -plugin install javascript     # JavaScript support
```

Plugin Configuration

Plugins can be configured in `~/.config/micro/settings.json`:

```
{
    "linter": true,
    "linter.ignoredMessages": [],
    "comment.leader": "# ",
    "filemanager.showdotfiles": true,
    "filemanager.showignored": false,
    "snippets.enabled": true,
    "autoclose.enabled": true
}
```

Real-World Usage Scenarios

Scenario 1: Linux System Configuration

```
# Edit SSH daemon configuration
sudo micro /etc/ssh/sshd_config

# Navigate to specific setting
Ctrl+F
# Type "PasswordAuthentication"
Enter

# Change the value
# Use mouse to click and position cursor
# Change "yes" to "no"

# Save and restart SSH service
Ctrl+S
Ctrl+Q
```

```
sudo systemctl restart sshd

# Alternative: Edit systemd service file
sudo micro /etc/systemd/system/myapp.service

# Add service configuration
[Unit]
Description=My Application
After=network.target

[Service]
Type=simple
User=myapp
ExecStart=/usr/local/bin/myapp
Restart=always

[Install]
WantedBy=multi-user.target

# Save and enable service
Ctrl+S
Ctrl+Q
sudo systemctl daemon-reload
sudo systemctl enable myapp
sudo systemctl start myapp
```

Scenario 2: Python Development

```
# Open Python file
micro script.py

# Enable Python-specific settings
# (Automatically detected by file extension)

# Multiple cursor editing for variable renaming
# Double-click on variable name
Alt+C  # Select all occurrences
# Type new variable name

# Comment/uncomment code blocks
# Select lines with mouse or Shift+arrows
Ctrl+/  # Toggle comments

# Duplicate lines for testing
Ctrl+D  # Duplicate current line
```

```
# Save file
Ctrl+S
```

Scenario 3: Log File Analysis

```
# Open log file
micro /var/log/application.log

# Search for errors
Ctrl+F
# Type "ERROR"
Enter

# Navigate through errors
Ctrl+N  # Next occurrence
Ctrl+P  # Previous occurrence

# Select error lines for copying
# Click and drag to select
Ctrl+C  # Copy

# Open new buffer for analysis
Ctrl+E
# Type "tab new"
Enter

# Paste errors
Ctrl+V

# Save analysis
Ctrl+S
# Type filename
Enter
```

Scenario 4: Web Development

```
# Open HTML file
micro index.html

# Use multiple cursors for tag editing
# Select opening tag
Alt+C  # Select all similar tags
# Edit all tags simultaneously
```

```
# Navigate between files
Ctrl+O
# Type filename or use arrow keys
Enter

# Split view for CSS editing
Ctrl+E
# Type "hsplit styles.css"
Enter

# Switch between splits
Ctrl+W  # Cycle through splits

# Format code (if formatter plugin installed)
Ctrl+E
# Type "format"
Enter
```

Advanced Features

Command Mode

Access command mode with `Ctrl+E`:

```
# File operations
open filename.txt       # Open file
save                    # Save current file
save filename.txt       # Save as
quit                    # Quit current buffer
exit                    # Exit micro

# Buffer operations
tab new                 # New tab
tab close               # Close current tab
tab switch N            # Switch to tab N
buffer N                # Switch to buffer N

# View operations
hsplit filename.txt     # Horizontal split
vsplit filename.txt     # Vertical split
unsplit                 # Close current split

# Search and replace
find pattern            # Find pattern
replace old new         # Replace old with new
```

```
replaceall old new    # Replace all occurrences

# Settings
set option value      # Set option
show option           # Show option value
reset option          # Reset option to default

# Plugin operations
plugin install name   # Install plugin
plugin remove name    # Remove plugin
plugin list           # List plugins
plugin update         # Update plugins
```

Macros

```
# Record macro
Ctrl+U  # Start recording
# Perform actions
Ctrl+U  # Stop recording

# Play macro
Ctrl+J  # Execute recorded macro
```

Bookmarks

```
# Set bookmark
Ctrl+B  # Toggle bookmark on current line

# Navigate bookmarks
Alt+Up    # Previous bookmark
Alt+Down  # Next bookmark

# List bookmarks
Ctrl+E
# Type "bookmark list"
Enter
```

Splits and Tabs

```
# Split operations
Ctrl+E hsplit filename  # Horizontal split
Ctrl+E vsplit filename  # Vertical split
Ctrl+W                  # Cycle through splits
```

```
Ctrl+Q                  # Close current split

# Tab operations
Ctrl+T                  # New tab
Ctrl+W                  # Close tab
Ctrl+Tab                # Next tab
Ctrl+Shift+Tab          # Previous tab
```

Integration with Development Tools

Git Integration

```
# View git diff
micro -option diffgutter true filename.txt

# Git commands through shell
Ctrl+E
# Type "term git status"
Enter

# Or use external git tools
# Configure git to use micro as editor
git config --global core.editor micro
```

Language Server Integration

While micro doesn't have built-in LSP support, you can use external tools:

```
# Install language servers
npm install -g typescript-language-server
pip install python-lsp-server

# Use with external LSP client
# Or use plugins that provide LSP integration
micro -plugin install lsp
```

Build System Integration

```
# Run build commands
Ctrl+E
# Type "term make"
Enter
```

```
# Or create custom commands
# In settings.json, add:
{
    "build": "make",
    "test": "npm test",
    "run": "python main.py"
}

# Then use:
Ctrl+E
# Type "build" or "test" or "run"
Enter
```

Customization Examples

Custom Status Line

```
{
    "statusformatl": "$(filename) [$(opt:filetype)] $(modified) | Line $(line)/$(lines) Col
    "statusformatr": "$(opt:encoding) | $(opt:fileformat) | Micro v$(version)"
}
```

Custom Key Bindings for Productivity

```
{
    "Ctrl+Shift+P": "lua:comment.comment",
    "Ctrl+D": "DuplicateLine",
    "Ctrl+Shift+K": "DeleteLine",
    "Ctrl+L": "SelectLine",
    "Ctrl+Shift+L": "SelectToEndOfLine",
    "F2": "Rename",
    "F5": "Reload",
    "F12": "ToggleHelp",
    "Ctrl+`": "ToggleTerminal"
}
```

File Type Specific Settings

Create `~/.config/micro/settings.json`:

```
{
    "*.py": {
        "tabsize": 4,
        "tabstospaces": true,
        "rmtrailingws": true
    },
    "*.js": {
        "tabsize": 2,
        "tabstospaces": true
    },
    "*.go": {
        "tabsize": 4,
        "tabstospaces": false
    },
    "*.md": {
        "softwrap": true,
        "wordwrap": true
    }
}
```

Performance and Optimization

Large File Handling

```
# Open large files efficiently
micro -option syntax false largefile.txt

# Disable features for better performance
micro -option hlsearch false
micro -option matchbrace false
micro -option autoindent false
```

Memory Usage

```
# Check memory usage
ps aux | grep micro

# Optimize for low memory
micro -option backup false
micro -option savehistory false
micro -option saveundo false
```

Troubleshooting

Common Issues

```
# Terminal compatibility issues
export TERM=xterm-256color
micro filename.txt

# Mouse not working
micro -option mouse true filename.txt

# Colors not displaying correctly
micro -option colorscheme default filename.txt

# Plugin issues
micro -plugin remove problematic-plugin
micro -plugin update
```

Debug Mode

```
# Enable debug logging
micro -debug filename.txt

# Check log file
tail -f ~/.config/micro/log.txt
```

Reset Configuration

```
# Backup current config
mv ~/.config/micro ~/.config/micro.backup

# Start fresh (micro will create new config)
micro
```

Migration from Other Editors

From Nano

Micro provides similar simplicity with more features: - Familiar Ctrl+S, Ctrl+O shortcuts - Mouse support - Better syntax highlighting - Plugin system

From GUI Editors

Micro bridges the gap between terminal and GUI editors: - Familiar keybindings (Ctrl+C, Ctrl+V, etc.) - Mouse support - Multiple cursors - Modern interface

From Vim

For Vim users wanting something simpler: - No modal editing - Intuitive keybindings - Built-in help system - Less configuration required

Micro excels as a user-friendly terminal editor that doesn't sacrifice power for simplicity. Its familiar interface, modern features, and extensibility make it an excellent choice for developers who want a capable editor without a steep learning curve.

Kakoune

Kakoune: The Selection-Based Editor

Kakoune is a modal editor that emphasizes selection-based editing and multiple selections. Inspired by Vim but with a different philosophy, Kakoune provides powerful text manipulation capabilities through its unique approach to editing.

What is Kakoune?

Kakoune (kak) is built around several key principles: - **Selection-first editing**: Select text first, then act on it - **Multiple selections**: Edit multiple locations simultaneously - **Orthogonal design**: Commands compose naturally - **Client-server architecture**: Multiple clients can connect to one session - **Powerful text objects**: Rich set of selection primitives - **Minimal configuration**: Works well with minimal setup

Why Choose Kakoune?

- **Intuitive workflow**: Selection-first approach feels natural
- **Multiple selections**: Powerful multi-cursor editing
- **Composable commands**: Commands work together logically
- **Fast and lightweight**: Efficient C++ implementation optimized for Linux
- **Extensible**: Scriptable with shell commands and Linux utilities
- **Unique approach**: Different from Vim/Emacs paradigms
- **Active community**: Growing ecosystem of plugins and tools
- **Linux-native**: Designed with Linux development workflows in mind

Installation

Package Managers

```
# Ubuntu/Debian
sudo apt update
sudo apt install kakoune

# Fedora
sudo dnf install kakoune

# Arch Linux
```

```
sudo pacman -S kakoune

# SUSE/openSUSE
sudo zypper install kakoune

# Gentoo
sudo emerge app-editors/kakoune

# Void Linux
sudo xbps-install kakoune

# FreeBSD
pkg install kakoune

# OpenBSD
pkg_add kakoune
```

Building from Source

```
# Install dependencies
# Ubuntu/Debian
sudo apt install build-essential libncurses5-dev libncursesw5-dev

# Clone and build
git clone https://github.com/mawww/kakoune.git
cd kakoune
make
sudo make install

# Or install to custom location
make install PREFIX=$HOME/.local
```

Development Version

```
# For latest features
git clone https://github.com/mawww/kakoune.git
cd kakoune
git checkout master
make
sudo make install
```

Basic Concepts

Selection-First Philosophy

Unlike Vim's action-first approach, Kakoune uses selection-first:

```
# Vim way: action + motion
dw    # Delete word (action first)

# Kakoune way: selection + action
w     # Select word
d     # Delete selection
```

Multiple Selections

Kakoune excels at multiple selections:

```
# Select all occurrences
%     # Select all matches of current selection
Alt+% # Select all matches in buffer

# Split selections
s     # Split selection on regex
Alt+s # Split selection on lines
```

Modes

Kakoune has several modes: - **Normal mode**: Default mode for navigation and commands - **Insert mode**: For typing text - **Prompt mode**: For entering commands and searches - **Menu mode**: For selecting from options

Essential Commands

Starting Kakoune

```
# Open file
kak filename.txt

# Open multiple files
kak file1.txt file2.txt

# Start server session
kak -s session_name filename.txt
```

```
# Connect to existing session
kak -c session_name

# List sessions
kak -l

# Kill session
kak -c session_name -e 'kill'
```

Basic Navigation

```
# Character movement
h     # Left
j     # Down
k     # Up
l     # Right

# Word movement
w     # Next word start
b     # Previous word start
e     # Next word end
Alt+w # Next WORD start
Alt+b # Previous WORD start
Alt+e # Next WORD end

# Line movement
0     # Line start
^     # First non-blank
$     # Line end
Alt+h # Line start (alternative)
Alt+l # Line end (alternative)

# Document movement
g g   # Document start
g e   # Document end
g k   # Buffer top
g j   # Buffer bottom

# Page movement
Ctrl+b # Page up
Ctrl+f # Page down
Ctrl+u # Half page up
Ctrl+d # Half page down
```

Text Selection

```
# Basic selection
x     # Select line
Alt+x # Extend selection to line
s     # Select regex matches
Alt+s # Split selection on lines

# Word selection
w     # Select word
W     # Select WORD
Alt+w # Extend to word
Alt+W # Extend to WORD

# Character selection
f<char> # Select to character
t<char> # Select until character
F<char> # Select to character (backward)
T<char> # Select until character (backward)

# Extend selection
Shift+<motion> # Extend selection with motion
;     # Reduce selection to cursor
Alt+; # Flip selection direction
```

Multiple Selections

```
# Create multiple selections
C     # Copy selection to new line
Alt+C # Copy selection to previous line
%     # Select all matches of current selection
Alt+% # Select all matches in buffer

# Manipulate selections
Alt+k # Keep selections matching regex
Alt+K # Keep selections not matching regex
$     # Pipe selections through command
|     # Pipe and replace selections
!     # Insert command output

# Selection iteration
)     # Rotate main selection forward
(     # Rotate main selection backward
Alt+) # Rotate selections forward
Alt+( # Rotate selections backward
```

Editing Commands

```
# Insert modes
i     # Insert before selection
a     # Insert after selection
I     # Insert at line start
A     # Insert at line end
o     # Insert new line below
O     # Insert new line above

# Delete/Change
d     # Delete selection
Alt+d # Delete to end of line
c     # Change selection (delete and insert)
Alt+c # Change to end of line

# Copy/Paste
y     # Yank (copy) selection
p     # Paste after selection
P     # Paste before selection
Alt+p # Paste all selections
R     # Replace selection with yanked text

# Case manipulation
~     # Switch case
`     # Convert to lowercase
Alt+` # Convert to uppercase

# Indentation
>     # Indent
<     # Unindent
Alt+> # Indent including empty lines
Alt+< # Unindent including empty lines
```

Search and Replace

```
# Search
/     # Search forward
?     # Search backward
n     # Next search result
N     # Previous search result (Alt+n)
*     # Search word under cursor

# Replace
r<char> # Replace character
```

```
s     # Select matches for replacement
c     # Change selection
|     # Pipe through sed for replacement

# Global replace example
%     # Select all
s pattern # Select all pattern matches
c replacement<Esc> # Replace all
```

File and Buffer Management

Buffer Operations

```
# Buffer navigation
g a   # Go to alternate buffer
g f   # Go to file under cursor
g F   # Go to file under cursor (new buffer)

# Buffer commands (in command mode)
:buffer <name>    # Switch to buffer
:buffer-next      # Next buffer
:buffer-previous  # Previous buffer
:delete-buffer    # Close current buffer
:write           # Save buffer
:write-all       # Save all buffers
:quit            # Quit
:quit!           # Quit without saving
:write-quit      # Save and quit
```

File Operations

```
# File commands
:edit <file>     # Open file
:new             # New buffer
:write <file>    # Save as file
:cd <dir>        # Change directory
:pwd             # Show current directory
```

Window Management

```
# Window splitting (requires tmux or similar)
# Kakoune doesn't have built-in splits
# Use tmux or terminal multiplexer

# Multiple clients
kak -s main file.txt    # Start session
kak -c main            # Connect new client
```

Configuration

Configuration File

Kakoune configuration is stored in `~/.config/kak/kakrc`:

```
# ~/.config/kak/kakrc

# Basic settings
set-option global tabstop 4
set-option global indentwidth 4
set-option global scrolloff 5
set-option global ui_options ncurses_status_on_top=true
set-option global ui_options ncurses_assistant=cat

# Enable line numbers
add-highlighter global/ number-lines -relative -hlcursor

# Enable matching bracket highlighting
add-highlighter global/ show-matching

# Enable whitespace highlighting
add-highlighter global/ show-whitespaces -tab "→" -tabpad "·" -lf "¬" -spc "·"

# Color scheme
colorscheme default

# Key mappings
map global normal <c-p> ': fzf-mode<ret>'
map global normal '#' ': comment-line<ret>'
map global normal '<a-#>' ': comment-block<ret>'

# User mode for custom commands
declare-user-mode user
map global normal <space> ': enter-user-mode user<ret>'
```

```
map global user f ': fzf-mode<ret>'
map global user g ': grep '
map global user r ': replace-mode<ret>'

# Auto-pairs
hook global InsertChar \( %{ exec '<a-;>)<left>' }
hook global InsertChar \[ %{ exec '<a-;>]<left>' }
hook global InsertChar \{ %{ exec '<a-;>}<left>' }
hook global InsertChar \" %{ exec '<a-;>"<left>' }
hook global InsertChar \' %{ exec '<a-;>''<left>' }

# Auto-indent
hook global InsertChar \n %{ exec -draft \; K <a-&> }
hook global InsertChar \} %{ exec -draft <a-;> <a-S> 1<a-&> }

# Highlight TODO/FIXME
add-highlighter global/ regex \b(TODO|FIXME|XXX|NOTE)\b 0:default+rb

# File type specific settings
hook global BufSetOption filetype=python %{
    set-option buffer indentwidth 4
    set-option buffer tabstop 4
}

hook global BufSetOption filetype=javascript %{
    set-option buffer indentwidth 2
    set-option buffer tabstop 2
}

hook global BufSetOption filetype=html %{
    set-option buffer indentwidth 2
    set-option buffer tabstop 2
}

# Clipboard integration
map global user y '<a-|>xclip -i -selection clipboard<ret>'
map global user p '!xclip -o -selection clipboard<ret>'

# Save and quit shortcuts
map global user w ': write<ret>'
map global user q ': quit<ret>'
map global user x ': write-quit<ret>'
```

Plugin Management

Kakoune doesn't have a built-in plugin manager, but you can use external tools:

```
# Install plug.kak (plugin manager)
mkdir -p ~/.config/kak/plugins
git clone https://github.com/andreyorst/plug.kak.git ~/.config/kak/plugins/plug.kak

# Add to kakrc
source "%val{config}/plugins/plug.kak/rc/plug.kak"
plug "andreyorst/plug.kak" noload

# Install plugins
plug "andreyorst/fzf.kak"
plug "andreyorst/smarttab.kak"
plug "ul/kak-lsp"
plug "alexherbo2/auto-pairs.kak"
plug "alexherbo2/surround.kak"
plug "delapouite/kakoune-buffers"
plug "occivink/kakoune-snippets"
```

Popular Plugins

```
# Essential plugins configuration

# FZF integration
plug "andreyorst/fzf.kak" config %{
    map global normal <c-p> ': fzf-mode<ret>'
    map global normal <c-o> ': fzf-file<ret>'
}

# LSP support
plug "ul/kak-lsp" do %{
    cargo install --locked --force --path .
} config %{
    set global lsp_diagnostic_line_error_sign ' '
    set global lsp_diagnostic_line_warning_sign ' '

    define-command ne -docstring 'go to next error/warning from lsp' %{
        lsp-find-error --include-warnings
    }
    define-command pe -docstring 'go to previous error/warning from lsp' %{
        lsp-find-error --previous --include-warnings
    }
    define-command ee -docstring 'go to current error/warning from lsp' %{
        lsp-find-error --include-warnings --previous
        lsp-find-error --include-warnings
    }
```

```
    map global user l %{: enter-user-mode lsp<ret>} -docstring "LSP mode"
    map global insert <tab> '<a-;>: try lsp-snippets-select-next-placeholders catch %{ execu
    map global object a '<a-semicolon>lsp-object<ret>' -docstring 'LSP any symbol'
    map global object <a-a> '<a-semicolon>lsp-object<ret>' -docstring 'LSP any symbol'
    map global object f '<a-semicolon>lsp-object Function Method<ret>' -docstring 'LSP funct
    map global object t '<a-semicolon>lsp-object Class Interface Struct<ret>' -docstring 'LS

    hook global WinSetOption filetype=(c|cpp|cc|rust|javascript|typescript|python) %{
        lsp-enable-window
    }
}

# Surround text objects
plug "alexherbo2/surround.kak" config %{
    map global user s ': surround<ret>' -docstring 'surround'
    map global user S ': surround _ _ * *<ret>' -docstring 'surround with *'
}

# Auto-pairs
plug "alexherbo2/auto-pairs.kak" config %{
    enable-auto-pairs
}

# Snippets
plug "occivink/kakoune-snippets" config %{
    set-option global snippets_directories "%opt{plug_install_dir}/kakoune-snippets/snippets
}

# Buffer management
plug "delapouite/kakoune-buffers" config %{
    map global user b ': enter-user-mode buffers<ret>' -docstring 'buffers'
    map global user B ': enter-user-mode -lock buffers<ret>' -docstring 'buffers (lock)'
}
```

Real-World Usage Scenarios

Scenario 1: Refactoring Variable Names

```
# Open file
kak script.py

# Select variable name
w     # Select word

# Select all occurrences
```

```
%     # Select all matches

# Change all occurrences
c     # Change selection
new_variable_name<Esc>

# Save file
:w
```

Scenario 2: Multiple Line Editing

```
# Select multiple lines
x     # Select line
j     # Move down
x     # Extend to next line
j     # Move down
x     # Extend to next line

# Or select lines with pattern
%     # Select all
s^def # Select lines starting with "def"

# Add prefix to all selected lines
I     # Insert at beginning
# Type prefix
<Esc>
```

Scenario 3: Complex Text Manipulation

```
# Select paragraph
<a-a>p # Select around paragraph

# Split into sentences
s\. # Split on periods

# Process each sentence
|capitalize_first_word.sh # Pipe through script

# Or use built-in commands
~     # Switch case of first character
```

Scenario 4: Log File Analysis

```
# Open log file
kak /var/log/application.log

# Select all error lines
%     # Select all
s^.*ERROR.*$ # Select lines containing ERROR

# Copy to new buffer
y     # Yank
:new  # New buffer
p     # Paste

# Analyze timestamps
s\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2} # Select timestamps
```

Advanced Features

Hooks and Automation

```
# Auto-save on focus lost
hook global FocusOut .* %{ write }

# Auto-format on save
hook global BufWritePre .*\.py %{ format }

# Highlight current word
hook global NormalIdle .* %{
    eval -draft %{ try %{
        exec <space><a-i>w <a-k>\A\w+\z<ret>
        add-highlighter window/word regex "\b\Q%val{selection}\E\b" 0:+u
    } catch %{
        rmhl window/word
    } }
}

# Auto-reload files changed externally
hook global FocusIn .* %{ edit! }
```

Custom Commands

```
# Define custom commands
define-command -docstring "duplicate line" duplicate-line %{
    exec 'x<a-s>y<a-o>p'
}

define-command -docstring "sort selections" sort-selections %{
    exec '|sort<ret>'
}

define-command -docstring "remove duplicates" remove-duplicates %{
    exec '|sort -u<ret>'
}

define-command -docstring "count words" count-words %{
    exec '<a-s>|wc -w<ret>'
}

# Map to keys
map global user d ': duplicate-line<ret>'
map global user s ': sort-selections<ret>'
map global user u ': remove-duplicates<ret>'
map global user c ': count-words<ret>'
```

Text Objects

```
# Built-in text objects
<a-i>w # Inner word
<a-a>w # Around word
<a-i>s # Inner sentence
<a-a>s # Around sentence
<a-i>p # Inner paragraph
<a-a>p # Around paragraph
<a-i>( # Inner parentheses
<a-a>( # Around parentheses
<a-i>{ # Inner braces
<a-a>{ # Around braces
<a-i>[ # Inner brackets
<a-a>[ # Around brackets
<a-i>" # Inner quotes
<a-a>" # Around quotes

# Custom text objects
define-command -docstring "select function" select-function %{
```

```
    exec '<a-s>s^def\s+\w+.*?(?=^def|\z)<ret>'
}

map global object f ': select-function<ret>'
```

Macros and Repetition

```
# Record macro
q<register> # Start recording to register
# ... perform actions ...
q           # Stop recording

# Play macro
@<register> # Execute macro from register
@@          # Repeat last macro

# Repeat last command
.           # Repeat last normal mode command
<a-.>       # Repeat last insert mode command
```

Language-Specific Configurations

Python Development

```
# Python-specific kakrc additions
hook global BufSetOption filetype=python %{
    set-option buffer indentwidth 4
    set-option buffer tabstop 4

    # Python-specific mappings
    map buffer user r ': python-run<ret>'
    map buffer user t ': python-test<ret>'
    map buffer user f ': python-format<ret>'

    # Auto-format with black
    hook buffer BufWritePre .* %{ exec '|black -<ret>' }

    # Highlight Python keywords
    add-highlighter buffer/python-keywords regex \b(def|class|import|from|if|elif|else|for|w
}

define-command python-run %{
    exec '|python3<ret>'
```

```
}

define-command python-test %{
    exec '!python3 -m pytest<ret>'
}

define-command python-format %{
    exec '%|black -<ret>'
}
```

Web Development

```
# HTML/CSS/JavaScript configuration
hook global BufSetOption filetype=(html|css|javascript) %{
    set-option buffer indentwidth 2
    set-option buffer tabstop 2

    # Auto-close tags for HTML
    hook buffer InsertChar > %{
        exec -draft '<a-;>h<a-k><[^/][^>]*<ret>a</<c-r>"><left>'
    }
}

# JavaScript-specific
hook global BufSetOption filetype=javascript %{
    map buffer user r ': javascript-run<ret>'
    map buffer user f ': javascript-format<ret>'

    # Format with prettier
    hook buffer BufWritePre .* %{ exec '%|prettier --stdin-filepath %val{buffile}<ret>' }
}

define-command javascript-run %{
    exec '|node<ret>'
}

define-command javascript-format %{
    exec '%|prettier --stdin-filepath %val{buffile}<ret>'
}
```

Go Development

```
# Go-specific configuration
hook global BufSetOption filetype=go %{
    set-option buffer indentwidth 4
    set-option buffer tabstop 4

    map buffer user r ': go-run<ret>'
    map buffer user b ': go-build<ret>'
    map buffer user t ': go-test<ret>'
    map buffer user f ': go-format<ret>'

    # Auto-format with gofmt
    hook buffer BufWritePre .* %{ exec '%|gofmt<ret>' }
}

define-command go-run %{
    exec '!go run %val{buffile}<ret>'
}

define-command go-build %{
    exec '!go build<ret>'
}

define-command go-test %{
    exec '!go test<ret>'
}

define-command go-format %{
    exec '%|gofmt<ret>'
}
```

Integration with External Tools

Git Integration

```
# Git commands
define-command git-status %{
    exec '!git status<ret>'
}

define-command git-add %{
    exec '!git add %val{buffile}<ret>'
}
```

```
define-command git-commit %{
    exec '!git commit<ret>'
}

define-command git-diff %{
    exec '!git diff %val{buffile}<ret>'
}

# Map to user mode
map global user g ': enter-user-mode git<ret>' -docstring 'git'
declare-user-mode git
map global git s ': git-status<ret>' -docstring 'status'
map global git a ': git-add<ret>' -docstring 'add'
map global git c ': git-commit<ret>' -docstring 'commit'
map global git d ': git-diff<ret>' -docstring 'diff'
```

Build System Integration

```
# Build system commands
define-command make %{
    exec '!make<ret>'
}

define-command make-clean %{
    exec '!make clean<ret>'
}

define-command make-install %{
    exec '!make install<ret>'
}

# Map to user mode
map global user m ': enter-user-mode make<ret>' -docstring 'make'
declare-user-mode make
map global make m ': make<ret>' -docstring 'make'
map global make c ': make-clean<ret>' -docstring 'clean'
map global make i ': make-install<ret>' -docstring 'install'
```

Terminal Integration

```
# Terminal commands
define-command terminal %{
    exec '!$SHELL<ret>'
}

define-command terminal-here %{
    exec '!cd %sh{dirname "$kak_buffile"} && $SHELL<ret>'
}

map global user t ': terminal<ret>' -docstring 'terminal'
map global user T ': terminal-here<ret>' -docstring 'terminal here'
```

Performance and Optimization

Large File Handling

```
# Optimize for large files
hook global BufCreate .*large.* %{
    set-option buffer readonly true
    rmhl buffer/number-lines
    rmhl buffer/show-matching
}

# Disable expensive features for large files
hook global BufOpenFile .* %{
    eval %sh{
        if [ $(wc -l < "$kak_hook_param") -gt 10000 ]; then
            echo "
                rmhl buffer/number-lines
                rmhl buffer/show-matching
                rmhl buffer/show-whitespaces
            "
        fi
    }
}
```

Memory Optimization

```
# Limit history size
set-option global history_size 100

# Disable undo for large operations
define-command no-undo %{
    set-option buffer history_size 0
}

map global user n ': no-undo<ret>' -docstring 'disable undo'
```

Troubleshooting

Common Issues

```
# Debug mode
kak -d

# Check configuration
:debug info

# Reset to defaults
kak -n  # No user config

# Check key mappings
:map

# Check options
:set-option
```

Session Management

```
# List sessions
kak -l

# Kill session
kak -c session_name -e 'kill'

# Kill all sessions
for session in $(kak -l); do
    kak -c "$session" -e 'kill'
done
```

Configuration Issues

```
# Test configuration
kak -f /dev/null  # Empty config

# Check for errors
kak -e 'echo %val{client_env_TERM}'

# Backup and reset config
mv ~/.config/kak ~/.config/kak.backup
```

Kakoune offers a unique and powerful approach to text editing through its selection-first philosophy and multiple selections. While it has a learning curve, its orthogonal design and composable commands make it highly effective for complex text manipulation tasks.

Modern Tools

Modern Linux Development Tools

This chapter covers modern command-line tools and utilities specifically designed to enhance the Linux development experience. These tools are optimized for Linux systems and integrate seamlessly with Linux workflows, providing powerful functionality that complements traditional editors.

Why Modern Tools on Linux?

- **Native Performance**: Tools written in Rust and Go provide excellent performance on Linux
- **System Integration**: Deep integration with Linux package managers and system services
- **Development Focus**: Designed with Linux development workflows in mind
- **Resource Efficiency**: Optimized for Linux server environments and containers
- **Open Source**: Most tools are open source and align with Linux philosophy

Terminal-Based Development Tools

Zellij: Modern Terminal Multiplexer

Zellij is a modern alternative to tmux with better defaults and user experience.

Installation

```
# Cargo
cargo install zellij

# Arch Linux
sudo pacman -S zellij

# Ubuntu/Debian (from GitHub releases)
wget https://github.com/zellij-org/zellij/releases/latest/download/zellij-x86_64-unknown-lin
tar -xzf zellij-x86_64-unknown-linux-musl.tar.gz
sudo mv zellij /usr/local/bin/

# SUSE/openSUSE
sudo zypper install zellij

# Gentoo
sudo emerge app-misc/zellij
```

Basic Usage

```
# Start zellij
zellij

# Key bindings (default)
Ctrl+p n    # New pane
Ctrl+p x    # Close pane
Ctrl+p h/j/k/l  # Navigate panes
Ctrl+p r    # Resize mode
Ctrl+p t    # New tab
Ctrl+p q    # Quit

# Sessions
zellij -s session_name    # Start named session
zellij ls                 # List sessions
zellij a session_name     # Attach to session
```

Configuration

```
// ~/.config/zellij/config.kdl
keybinds {
    normal {
        bind "Alt h" { MoveFocus "Left"; }
        bind "Alt l" { MoveFocus "Right"; }
        bind "Alt j" { MoveFocus "Down"; }
        bind "Alt k" { MoveFocus "Up"; }
        bind "Alt n" { NewPane; }
        bind "Alt x" { CloseFocus; }
        bind "Alt t" { NewTab; }
        bind "Alt 1" { GoToTab 1; }
        bind "Alt 2" { GoToTab 2; }
        bind "Alt 3" { GoToTab 3; }
    }
}

theme "catppuccin-mocha"
default_shell "zsh"
pane_frames false
```

Starship: Cross-Shell Prompt

Starship provides a fast, customizable prompt for any shell.

Installation

```
# Install script
curl -sS https://starship.rs/install.sh | sh

# Cargo
cargo install starship --locked

# Package managers
# Arch Linux
sudo pacman -S starship

# Ubuntu/Debian
sudo apt install starship

# SUSE/openSUSE
sudo zypper install starship

# Gentoo
sudo emerge app-shells/starship
```

Configuration

```
# Add to shell config (~/.bashrc, ~/.zshrc, etc.)
eval "$(starship init bash)"  # For bash
eval "$(starship init zsh)"   # For zsh
```

```
# ~/.config/starship.toml
format = """
[](#9A348E)\
$os\
$username\
[](bg:#DA627D fg:#9A348E)\
$directory\
[](fg:#DA627D bg:#FCA17D)\
$git_branch\
$git_status\
[](fg:#FCA17D bg:#86BBD8)\
$c\
$elixir\
$elm\
$golang\
$gradle\
$haskell\
```

```
$java\
$julia\
$nodejs\
$nim\
$rust\
$scala\
[](fg:#86BBD8 bg:#06969A)\
$docker_context\
[](fg:#06969A bg:#33658A)\
$time\
[ ](fg:#33658A)\
"""

[os]
disabled = false
style = "bg:#9A348E"

[username]
show_always = true
style_user = "bg:#9A348E"
style_root = "bg:#9A348E"
format = '[$user ]($style)'
disabled = false

[directory]
style = "bg:#DA627D"
format = "[ $path ]($style)"
truncation_length = 3
truncation_symbol = "…/"

[git_branch]
symbol = ""
style = "bg:#FCA17D"
format = '[ $symbol $branch ]($style)'

[git_status]
style = "bg:#FCA17D"
format = '[$all_status$ahead_behind ]($style)'

[nodejs]
symbol = ""
style = "bg:#86BBD8"
format = '[ $symbol ($version) ]($style)'

[rust]
symbol = ""
```

```
style = "bg:#86BBD8"
format = '[ $symbol ($version) ]($style)'

[golang]
symbol = ""
style = "bg:#86BBD8"
format = '[ $symbol ($version) ]($style)'

[python]
symbol = ""
style = "bg:#86BBD8"
format = '[ $symbol ($version) ]($style)'

[time]
disabled = false
time_format = "%R"
style = "bg:#33658A"
format = '[  $time ]($style)'
```

Zoxide: Smart Directory Navigation

Zoxide is a smarter cd command that learns your habits.

Installation

```
# Install script
curl -sS https://raw.githubusercontent.com/ajeetdsouza/zoxide/main/install.sh | bash

# Cargo
cargo install zoxide --locked

# Package managers
sudo pacman -S zoxide  # Arch
sudo zypper install zoxide  # SUSE/openSUSE
sudo emerge app-shells/zoxide  # Gentoo
```

Setup

```
# Add to shell config
eval "$(zoxide init bash)"  # For bash
eval "$(zoxide init zsh)"   # For zsh
```

Usage

```
# Navigate to directories
z foo       # cd to highest ranked directory matching foo
z foo bar   # cd to highest ranked directory matching foo and bar
zi foo      # cd with interactive selection

# Add directories
zoxide add /path/to/directory

# Query database
zoxide query foo
zoxide query --list
```

Eza: Modern ls Replacement

Eza is a modern replacement for ls with better defaults and features.

Installation

```
# Cargo
cargo install eza

# Arch Linux
sudo pacman -S eza

# Ubuntu/Debian
sudo apt install eza

# SUSE/openSUSE
sudo zypper install eza

# Gentoo
sudo emerge sys-apps/eza
```

Usage

```
# Basic usage
eza                     # List files
eza -l                  # Long format
eza -la                 # Long format with hidden files
eza -T                  # Tree view
eza -T -L 2             # Tree view with depth limit
```

```
eza --git              # Show git status
eza --icons            # Show file type icons

# Useful aliases
alias ls='eza'
alias ll='eza -l'
alias la='eza -la'
alias lt='eza -T'
alias lg='eza -l --git'
```

Bat: Better Cat

Bat is a cat clone with syntax highlighting and Git integration.

Installation

```
# Package managers
sudo apt install bat        # Ubuntu/Debian
sudo pacman -S bat          # Arch Linux
sudo zypper install bat     # SUSE/openSUSE
sudo emerge sys-apps/bat    # Gentoo
cargo install bat           # Cargo
```

Usage

```
# Basic usage
bat file.txt               # View file with syntax highlighting
bat -n file.txt            # Show line numbers
bat -A file.txt            # Show all characters
bat --theme=GitHub file.txt # Use specific theme

# Useful aliases
alias cat='bat'
alias less='bat'

# Integration with other tools
export MANPAGER="sh -c 'col -bx | bat -l man -p'"
```

Ripgrep: Fast Text Search

Ripgrep (rg) is a fast grep alternative with better defaults.

Installation

```
# Package managers
sudo apt install ripgrep    # Ubuntu/Debian
sudo pacman -S ripgrep      # Arch Linux
sudo zypper install ripgrep # SUSE/openSUSE
sudo emerge sys-apps/ripgrep # Gentoo
cargo install ripgrep       # Cargo
```

Usage

```
# Basic search
rg "pattern"               # Search in current directory
rg "pattern" /path         # Search in specific path
rg -i "pattern"            # Case insensitive
rg -w "word"               # Whole word search
rg -v "pattern"            # Invert match

# File type filtering
rg -t py "pattern"         # Search only Python files
rg -T py "pattern"         # Exclude Python files
rg --type-list             # List available file types

# Advanced options
rg -n "pattern"            # Show line numbers
rg -C 3 "pattern"          # Show 3 lines of context
rg -A 3 -B 3 "pattern"     # Show 3 lines after and before
rg --hidden "pattern"      # Search hidden files
rg --no-ignore "pattern"   # Don't respect .gitignore

# Regular expressions
rg "\d{3}-\d{2}-\d{4}"     # Phone number pattern
rg "^function.*\{$"        # Function definitions
```

Fd: Better Find

Fd is a simple, fast alternative to find with better defaults.

Installation

```
# Package managers
sudo apt install fd-find   # Ubuntu/Debian (command is fdfind)
sudo pacman -S fd          # Arch Linux
sudo zypper install fd     # SUSE/openSUSE
sudo emerge sys-apps/fd    # Gentoo
cargo install fd-find      # Cargo
```

Usage

```
# Basic usage
fd pattern                 # Find files/directories matching pattern
fd -t f pattern            # Find only files
fd -t d pattern            # Find only directories
fd -e py                   # Find Python files
fd -E node_modules pattern # Exclude node_modules

# Advanced options
fd -H pattern              # Include hidden files
fd -I pattern              # Don't respect .gitignore
fd -x command              # Execute command on results
fd pattern -x ls -la       # Execute ls -la on each result

# Useful aliases
alias find='fd'
```

Delta: Better Git Diff

Delta provides syntax highlighting and better formatting for git diffs.

Installation

```
# Package managers
sudo apt install git-delta # Ubuntu/Debian
sudo pacman -S git-delta   # Arch Linux
sudo zypper install git-delta # SUSE/openSUSE
sudo emerge dev-util/git-delta # Gentoo
cargo install git-delta    # Cargo
```

Configuration

```
# Add to ~/.gitconfig
[core]
    pager = delta

[interactive]
    diffFilter = delta --color-only

[delta]
    navigate = true
    light = false
    side-by-side = true
    line-numbers = true
    syntax-theme = Dracula

[merge]
    conflictstyle = diff3

[diff]
    colorMoved = default
```

File Management Tools

Broot: Interactive Directory Navigation

Broot provides an interactive way to navigate and manipulate directory trees.

Installation

```
# Cargo
cargo install broot

# Package managers
sudo pacman -S broot       # Arch Linux
sudo zypper install broot  # SUSE/openSUSE
sudo emerge app-misc/broot # Gentoo

# Install shell integration
broot --install
```

Usage

```
# Start broot
br                          # Navigate current directory
br /path/to/directory       # Navigate specific directory

# Key bindings
Ctrl+h                      # Toggle hidden files
Ctrl+f                      # Search files
Ctrl+d                      # Delete file/directory
Ctrl+c                      # Copy path
Enter                       # Open file/enter directory
Alt+Enter                   # Open in external editor
```

Nnn: Terminal File Manager

Nnn is a fast, feature-rich terminal file manager.

Installation

```
# Package managers
sudo apt install nnn        # Ubuntu/Debian
sudo pacman -S nnn          # Arch Linux
sudo zypper install nnn     # SUSE/openSUSE
sudo emerge app-misc/nnn    # Gentoo

# From source
git clone https://github.com/jarun/nnn.git
cd nnn
make
sudo make install
```

Configuration

```
# Environment variables
export NNN_PLUG='f:finder;o:fzopen;p:mocplay;d:diffs;t:nmount;v:imgview'
export NNN_FIFO='/tmp/nnn.fifo'
export NNN_COLORS='2136'
export NNN_FCOLORS='c1e2272e006033f7c6d6abc4'

# Aliases
alias n='nnn -de'
alias N='sudo nnn -dH'
```

Usage

```
# Basic navigation
j/k or ↑/↓                # Navigate files
h/l or ←/→                # Navigate directories
Enter                     # Open file/enter directory
q                         # Quit
?                         # Help

# File operations
d                         # Delete
r                         # Rename
c                         # Copy
v                         # Move
s                         # Select files
a                         # Select all
A                         # Invert selection

# Advanced features
;                         # Plugin menu
!                         # Shell
e                         # Edit in $EDITOR
p                         # Show file info
```

Development Environment Tools

Direnv: Environment Management

Direnv automatically loads and unloads environment variables based on directory.

Installation

```
# Package managers
sudo apt install direnv    # Ubuntu/Debian
sudo pacman -S direnv      # Arch Linux
sudo zypper install direnv # SUSE/openSUSE
sudo emerge dev-util/direnv # Gentoo

# Add to shell config
eval "$(direnv hook bash)" # For bash
eval "$(direnv hook zsh)"  # For zsh
```

Usage

```
# Create .envrc file in project directory
echo 'export DATABASE_URL="postgres://localhost/mydb"' > .envrc
echo 'export DEBUG=true' >> .envrc

# Allow the .envrc file
direnv allow

# The environment variables are now loaded when you cd into the directory
# and unloaded when you leave
```

Advanced .envrc Examples

```
# Python virtual environment
layout python3
pip install -r requirements.txt

# Node.js version management
use node 18

# Go module
export GOPATH=$PWD/.go
export PATH=$GOPATH/bin:$PATH

# Docker development
export COMPOSE_FILE=docker-compose.dev.yml
export DOCKER_BUILDKIT=1

# Load from .env file
dotenv

# Path manipulation
PATH_add bin
PATH_add scripts
```

Mise: Runtime Version Manager

Mise (formerly rtx) manages multiple runtime versions for different languages.

Installation

```
# Install script
curl https://mise.run | sh

# Cargo
cargo install mise

# Package managers
sudo pacman -S mise        # Arch Linux
sudo zypper install mise   # SUSE/openSUSE
# For Gentoo, install via cargo or from source
```

Setup

```
# Add to shell config
echo 'eval "$(mise activate bash)"' >> ~/.bashrc  # For bash
echo 'eval "$(mise activate zsh)"' >> ~/.zshrc    # For zsh
```

Usage

```
# Install runtimes
mise install node@18       # Install Node.js 18
mise install python@3.11   # Install Python 3.11
mise install go@1.21       # Install Go 1.21

# Set global versions
mise global node@18
mise global python@3.11

# Set local versions (creates .mise.toml)
mise local node@16
mise local python@3.9

# List available versions
mise list-remote node
mise list-remote python

# List installed versions
mise list
```

Configuration

```
# .mise.toml
[tools]
node = "18"
python = "3.11"
go = "1.21"

[env]
NODE_ENV = "development"
DEBUG = "true"

[tasks.dev]
run = "npm run dev"
description = "Start development server"

[tasks.test]
run = "npm test"
description = "Run tests"
```

Just: Command Runner

Just is a command runner similar to make but with better syntax.

Installation

```
# Cargo
cargo install just

# Package managers
sudo pacman -S just        # Arch Linux
sudo zypper install just   # SUSE/openSUSE
sudo emerge sys-devel/just # Gentoo

# Install script
curl --proto '=https' --tlsv1.2 -sSf https://just.systems/install.sh | bash -s -- --to ~/bin
```

Usage

Create a `justfile`:

```
# Default recipe
default:
```

```
    @just --list

# Development server
dev:
    npm run dev

# Run tests
test:
    npm test

# Build project
build:
    npm run build

# Deploy to production
deploy: build
    rsync -av dist/ user@server:/var/www/

# Clean build artifacts
clean:
    rm -rf dist/
    rm -rf node_modules/

# Install dependencies
install:
    npm install

# Format code
fmt:
    prettier --write .
    eslint --fix .

# Docker commands
docker-build:
    docker build -t myapp .

docker-run:
    docker run -p 3000:3000 myapp

# Database commands
db-migrate:
    npx prisma migrate dev

db-seed:
    npx prisma db seed

# Variables
```

```
port := "3000"
env := "development"

# Recipe with parameters
serve port=port:
    NODE_ENV={{env}} npm start -- --port {{port}}

# Conditional recipes
backup:
    #!/usr/bin/env bash
    if [ -f "database.db" ]; then
        cp database.db "database.db.backup.$(date +%Y%m%d_%H%M%S)"
        echo "Database backed up"
    else
        echo "No database file found"
    fi
```

```
# Run recipes
just                    # Run default recipe
just dev                # Run dev recipe
just serve 8080         # Run serve with custom port
just --list             # List all recipes
```

Text Processing Tools

Jq: JSON Processor

Jq is a lightweight command-line JSON processor.

Installation

```
# Package managers
sudo apt install jq       # Ubuntu/Debian
sudo pacman -S jq         # Arch Linux
sudo zypper install jq    # SUSE/openSUSE
sudo emerge app-misc/jq   # Gentoo
```

Usage

```
# Basic usage
echo '{"name": "John", "age": 30}' | jq '.'
echo '{"name": "John", "age": 30}' | jq '.name'
echo '[{"name": "John"}, {"name": "Jane"}]' | jq '.[0].name'

# Filter arrays
echo '[1,2,3,4,5]' | jq '.[] | select(. > 3)'

# Transform data
echo '{"users": [{"name": "John", "age": 30}, {"name": "Jane", "age": 25}]}' | \
  jq '.users | map({name, adult: (.age >= 18)})'

# Real-world examples
curl -s https://api.github.com/users/octocat | jq '.name, .public_repos'
docker inspect container_name | jq '.[0].NetworkSettings.IPAddress'
kubectl get pods -o json | jq '.items[].metadata.name'
```

Yq: YAML Processor

Yq is like jq but for YAML files.

Installation

```
# Go version (recommended)
go install github.com/mikefarah/yq/v4@latest

# Package managers
sudo apt install yq       # Ubuntu/Debian
sudo zypper install yq    # SUSE/openSUSE
sudo emerge app-misc/yq   # Gentoo
```

Usage

```
# Basic usage
yq '.name' config.yaml
yq '.database.host' config.yaml
yq '.services[0].name' docker-compose.yml

# Modify YAML
yq '.database.port = 5432' config.yaml
yq '.services[0].image = "nginx:latest"' docker-compose.yml
```

```
# Multiple files
yq eval-all '. as $item ireduce ({}; . * $item)' file1.yaml file2.yaml
```

Sd: Sed Alternative

Sd is a more intuitive alternative to sed for find and replace operations.

Installation

```
# Cargo
cargo install sd

# Package managers
sudo pacman -S sd          # Arch Linux
sudo zypper install sd     # SUSE/openSUSE
sudo emerge sys-apps/sd    # Gentoo
```

Usage

```
# Basic find and replace
sd 'old_text' 'new_text' file.txt
sd '\d{4}-\d{2}-\d{2}' 'DATE' file.txt

# In-place editing
sd -i 'old_text' 'new_text' file.txt

# Preview changes
sd 'old_text' 'new_text' file.txt --preview

# Multiple files
sd 'old_text' 'new_text' *.txt
```

Monitoring and System Tools

Htop: Process Viewer

Htop is an interactive process viewer and system monitor.

Installation

```
# Package managers
sudo apt install htop     # Ubuntu/Debian
sudo pacman -S htop       # Arch Linux
sudo zypper install htop  # SUSE/openSUSE
sudo emerge sys-process/htop # Gentoo
```

Usage

```
# Start htop
htop

# Key bindings
F1                        # Help
F2                        # Setup
F3                        # Search
F4                        # Filter
F5                        # Tree view
F6                        # Sort by
F9                        # Kill process
F10                       # Quit
```

Bottom: System Monitor

Bottom (btm) is a cross-platform system monitor written in Rust.

Installation

```
# Cargo
cargo install bottom

# Package managers
sudo pacman -S bottom     # Arch Linux
sudo zypper install bottom # SUSE/openSUSE
sudo emerge sys-process/bottom # Gentoo

# Debian/Ubuntu
curl -LO https://github.com/ClementTsang/bottom/releases/download/0.9.6/bottom_0.9.6_amd64.d
sudo dpkg -i bottom_0.9.6_amd64.deb
```

Usage

```
# Start bottom
btm

# Key bindings
?                       # Help
q                       # Quit
/                       # Search
Tab                     # Next widget
Shift+Tab               # Previous widget
+/-                     # Zoom in/out
```

Procs: Process Information

Procs is a modern replacement for ps with better output formatting.

Installation

```
# Cargo
cargo install procs

# Package managers
sudo pacman -S procs      # Arch Linux
sudo zypper install procs # SUSE/openSUSE
sudo emerge sys-process/procs # Gentoo
```

Usage

```
# Basic usage
procs                     # List all processes
procs firefox             # Find processes matching "firefox"
procs --tree              # Tree view
procs --watch             # Watch mode
procs --sortd cpu         # Sort by CPU usage (descending)
```

Linux-Specific Integration

Systemd Integration

Many modern tools can be integrated with systemd for better Linux system management:

```
# Create user service for auto-updating tools
mkdir -p ~/.config/systemd/user

# Auto-update mise tools
cat > ~/.config/systemd/user/mise-update.service << 'EOF'
[Unit]
Description=Update mise tools
After=network.target

[Service]
Type=oneshot
ExecStart=/usr/local/bin/mise upgrade
User=%i
EOF

cat > ~/.config/systemd/user/mise-update.timer << 'EOF'
[Unit]
Description=Update mise tools daily

[Timer]
OnCalendar=daily
Persistent=true

[Install]
WantedBy=timers.target
EOF

# Enable the timer
systemctl --user enable mise-update.timer
systemctl --user start mise-update.timer
```

Linux Distribution Package Management

```
# Create a script to install modern tools across different distros
#!/bin/bash
# install-modern-tools.sh

detect_distro() {
    if [ -f /etc/os-release ]; then
        . /etc/os-release
        echo $ID
    elif [ -f /etc/redhat-release ]; then
        echo "rhel"
    elif [ -f /etc/debian_version ]; then
        echo "debian"
```

```
    else
        echo "unknown"
    fi
}

install_tools() {
    local distro=$(detect_distro)

    case $distro in
        ubuntu|debian)
            sudo apt update
            sudo apt install -y ripgrep fd-find bat eza zoxide starship direnv
            ;;
        arch|manjaro)
            sudo pacman -S ripgrep fd bat eza zoxide starship direnv
            ;;
        fedora)
            sudo dnf install -y ripgrep fd-find bat eza zoxide starship direnv
            ;;
        opensuse*|suse)
            sudo zypper install -y ripgrep fd bat eza zoxide starship direnv
            ;;
        gentoo)
            sudo emerge sys-apps/ripgrep sys-apps/fd sys-apps/bat sys-apps/eza \
                        app-shells/zoxide app-shells/starship dev-util/direnv
            ;;
        *)
            echo "Unsupported distribution: $distro"
            echo "Please install tools manually using cargo or from source"
            ;;
    esac
}

install_tools
```

Linux Desktop Integration

```
# Create desktop entries for terminal-based tools
mkdir -p ~/.local/share/applications

# Helix desktop entry
cat > ~/.local/share/applications/helix.desktop << 'EOF'
[Desktop Entry]
Name=Helix
Comment=A post-modern modal text editor
```

```
Exec=gnome-terminal -- hx %F
Icon=text-editor
Terminal=false
Type=Application
Categories=Development;TextEditor;
MimeType=text/plain;text/x-chdr;text/x-csrc;text/x-c++hdr;text/x-c++src;text/x-java;text/x-d
EOF

# Micro desktop entry
cat > ~/.local/share/applications/micro.desktop << 'EOF'
[Desktop Entry]
Name=Micro
Comment=A modern and intuitive terminal-based text editor
Exec=gnome-terminal -- micro %F
Icon=text-editor
Terminal=false
Type=Application
Categories=Development;TextEditor;
MimeType=text/plain;text/x-chdr;text/x-csrc;text/x-c++hdr;text/x-c++src;text/x-java;text/x-d
EOF

# Update desktop database
update-desktop-database ~/.local/share/applications
```

Integration Examples

Complete Development Setup

```
# ~/.zshrc or ~/.bashrc

# Modern tools
eval "$(starship init zsh)"
eval "$(zoxide init zsh)"
eval "$(direnv hook zsh)"
eval "$(mise activate zsh)"

# Aliases
alias ls='eza'
alias ll='eza -l'
alias la='eza -la'
alias lt='eza -T'
alias cat='bat'
alias find='fd'
alias grep='rg'
```

```
alias ps='procs'
alias top='btm'

# Functions
function fzf-edit() {
    local file
    file=$(fd --type f | fzf --preview 'bat --color=always {}')
    if [[ -n $file ]]; then
        $EDITOR "$file"
    fi
}

function fzf-cd() {
    local dir
    dir=$(fd --type d | fzf --preview 'eza -T {} | head -50')
    if [[ -n $dir ]]; then
        cd "$dir"
    fi
}

# Key bindings
bindkey '^F' fzf-edit
bindkey '^G' fzf-cd
```

Project Template with Modern Tools

```
# project-setup.sh
#!/bin/bash

PROJECT_NAME=$1
if [[ -z $PROJECT_NAME ]]; then
    echo "Usage: $0 <project-name>"
    exit 1
fi

# Create project directory
mkdir "$PROJECT_NAME"
cd "$PROJECT_NAME"

# Initialize git
git init

# Create .envrc for direnv
cat > .envrc << 'EOF'
export PROJECT_NAME=$(basename $PWD)
```

```
export NODE_ENV=development
export DEBUG=true

# Add project bin to PATH
PATH_add bin
PATH_add scripts

# Load .env if it exists
dotenv_if_exists
EOF

# Create .mise.toml for runtime management
cat > .mise.toml << 'EOF'
[tools]
node = "18"
python = "3.11"

[env]
NODE_ENV = "development"

[tasks.dev]
run = "npm run dev"
description = "Start development server"

[tasks.test]
run = "npm test"
description = "Run tests"

[tasks.build]
run = "npm run build"
description = "Build project"
EOF

# Create justfile for task running
cat > justfile << 'EOF'
# List available recipes
default:
    @just --list

# Install dependencies
install:
    npm install

# Start development server
dev:
    npm run dev
```

```
# Run tests
test:
    npm test

# Build project
build:
    npm run build

# Format code
fmt:
    prettier --write .
    eslint --fix .

# Clean build artifacts
clean:
    rm -rf dist/
    rm -rf node_modules/
EOF

# Create .gitignore
cat > .gitignore << 'EOF'
node_modules/
dist/
.env
.DS_Store
*.log
EOF

# Allow direnv
direnv allow

echo "Project $PROJECT_NAME created successfully!"
echo "Run 'cd $PROJECT_NAME && just install' to get started"
```

Linux-Specific Troubleshooting

Permission Issues

```
# Fix common permission issues with modern tools
# Ensure tools are executable
chmod +x ~/.local/bin/*

# Fix cargo-installed tools permissions
chmod +x ~/.cargo/bin/*
```

```
# Add to PATH if needed
echo 'export PATH="$HOME/.local/bin:$HOME/.cargo/bin:$PATH"' >> ~/.bashrc
```

Package Manager Conflicts

```
# Handle conflicts between package managers
# Remove system package before installing via cargo
sudo apt remove ripgrep  # Remove system version
cargo install ripgrep    # Install latest via cargo

# Or use alternatives command on Debian/Ubuntu
sudo update-alternatives --install /usr/bin/rg rg /usr/bin/rg 10
sudo update-alternatives --install /usr/bin/rg rg ~/.cargo/bin/rg 20
```

Performance Optimization for Linux

```
# Optimize tools for Linux systems
# Set environment variables for better performance
export RIPGREP_CONFIG_PATH="$HOME/.ripgreprc"
export BAT_THEME="Dracula"
export FZF_DEFAULT_COMMAND='fd --type f --hidden --follow --exclude .git'

# Create ripgrep config for Linux development
cat > ~/.ripgreprc << 'EOF'
# Search hidden files but not .git
--hidden
--glob=!.git/*

# Follow symbolic links
--follow

# Smart case search
--smart-case

# Show line numbers
--line-number

# Limit search depth for performance
--max-depth=10
EOF
```

Container Integration

```
# Use modern tools in Docker containers
# Dockerfile example
FROM ubuntu:22.04

RUN apt-get update && apt-get install -y \
    curl \
    git \
    build-essential \
    && rm -rf /var/lib/apt/lists/*

# Install Rust for cargo-based tools
RUN curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y
ENV PATH="/root/.cargo/bin:${PATH}"

# Install modern tools
RUN cargo install ripgrep fd-find bat eza zoxide starship

# Set up shell with modern tools
RUN echo 'eval "$(starship init bash)"' >> ~/.bashrc
RUN echo 'eval "$(zoxide init bash)"' >> ~/.bashrc
RUN echo 'alias ls=eza' >> ~/.bashrc
RUN echo 'alias cat=bat' >> ~/.bashrc
```

These modern tools significantly enhance the Linux development experience by providing better defaults, improved performance, and more intuitive interfaces compared to traditional Unix tools. They integrate seamlessly with Linux systems and are optimized for Linux development workflows, making them essential tools for modern Linux developers and system administrators.