Configure the Command Line for Data Science Development

Setting up Kitty terminal, Zsh shell, and productivity tools for efficient data science workflows

Ronald Glenn Thomas

2025-07-24

Table of contents

Figure 1: UCSD Geisel Library - A hub for research and academic discovery

The Geisel Library at UC San Diego, where research and innovation converge - a perfect setting for learning advanced terminal configuration techniques that enhance data science productivity

1 Introduction

The command line is a fundamental tool for data science development, offering powerful capabilities that graphical interfaces cannot match. A well-configured terminal environment can dramatically improve your productivity, workflow efficiency, and development experience.

This guide focuses on setting up Kitty terminal and Zsh shell with productivity-enhancing plugins, custom aliases, and optimized configurations for data science work on both macOS and Linux systems.

By the end of this post, you'll be able to:

- Install and configure Kitty terminal with optimal settings for data science
- Set up Zsh with productivity plugins and custom aliases
- Implement efficient navigation and file management workflows
- Configure environment variables and exports for development tools
- Create custom functions for common data science tasks

2 Prerequisites and Setup

Before we begin, ensure you have the following:

Required Tools: - macOS: Homebrew package manager - Linux: Package manager (apt, yum, etc.) - Administrative privileges for software installation

Installation Steps:

On both macOS and Linux, we will be using Kitty as our terminal emulator and Zsh as our shell.

macOS Installation:

```
brew install kitty
```

Linux Installation:

```
# Ubuntu/Debian
sudo apt-get install kitty

# Fedora/RHEL
sudo dnf install kitty

# Arch Linux
sudo pacman -S kitty
```

There are a huge number of configuration options for kitty. Here are the few that recommened to change for usability purposes.

```
# ~/.config/kitty/kitty.conf

map ctrl+shift+z       toggle_layout stack
draw_minimal_borders yes
background_opacity 0.99
font_family     FiraCode Nerd Font Mono
font_size       14.0
window_margin_width    5
background_image        /Users/zenn/docs/backgrounds/NxyVIMp.jpeg
window_border_width     1.5pt
include current-theme.conf
```

3 Notes on setting up iterm2:

Useful Keyboard shortcuts

```
cmd + shift + enter -> maximize current window (especially useful when you use split windows)
cmd + t -> open a new tab
cmd + w -> close focused window
cmd + d -> split window vertically
cmd + shift + d -> split window horizontally
cmd + {1..9} -> Move to tab with number {1..9}
cmd + i -> change name of the tab
```

Set up the status bar for time, git, directory. This reduces the burden on the shell prompt.

settings/profiles/session/

check “status bar enabled” at bottom of screen

In interactive screen drag “current directory”, “Empty space”, and “git state” into Active components section.

settings/appearance

“Status Bar location” select “Bottom”

or alternatively

cd ~

ln -s ~/Dropbox/dotfiles/iterm2Profiles.json .

in iterm2 profiles/Other Actions/Import JSON profiles iterm2Profiles.json

Set out the configuration needs for the shell

Good place to start:

discuss plugins particularly

z vs scd vs wd

4 .zshrc

```
PS1='%B%2~ %(?:%F{green} %f:%F{red} %f)%b '
setopt auto_cd auto_pushd pushd_ignore_dups pushdminus
bindkey -v
autoload -U compinit && compinit -u && compinit && compdef _dirs d
source /opt/homebrew/etc/profile.d/autojump.sh
source ~/.iterm2_shell_integration.zsh
source /opt/homebrew/share/zsh-syntax-highlighting/zsh-syntax-highlighting.zsh
source  /opt/homebrew/share/zsh-autosuggestions/zsh-autosuggestions.zsh
source ~/.config/zsh/.zsh_aliases
source ~/.config/zsh/.zsh_functions
source ~/.config/zsh/.zsh_exports
```

4.1 ~/.config/zsh/.zsh_aliases

```
alias mm='mutt'
alias sk='open -a Skim'
alias vc='vim ~/.vimrc'
alias vz='vim ~/.zshrc'
alias sz='source ~/.zshrc'
alias p2='enscript -C -2 -r -j --media=Letter'
alias p1='enscript  -j --media=Letter'
alias yr="yabai --restart-service"
alias lt='eza -lrFha -sold'
alias mvim="/Applications/MacVim.app/Contents/bin/mvim"
alias tp='trash-put -v'
alias rm='echo "This is not the command you are looking for."; false'
alias s='scd'
```

```
alias ZZ='exit'
alias r="radian"
alias nt="nvim"
alias -g ...='../..'
alias -g ....='../../..'
alias -g .....='../../../..'
alias -g ......='../../../../..'

alias -- -='cd -'
alias 1='cd -1'
alias 2='cd -2'
alias 3='cd -3'
alias 4='cd -4'
alias 5='cd -5'
alias 6='cd -6'
alias 7='cd -7'
alias 8='cd -8'
alias 9='cd -9'

alias md='mkdir -p'
alias rd=rmdir

# List directory contents
alias lsa='ls -lah'
alias l='ls -lah'
alias ll='ls -lh'
alias la='ls -lAh'

# search for directory and cd to it
alias sd="cd ~ && cd \$(find * -type d -not -path '*/Library/*' | fzf)"
```

4.2 ~/.config/zsh/.zsh_exports

```
export EDITOR="vim"
export TEXINPUTS='.:/Users/zenn/shr/images:/Users/zenn/shr:'
export PATH=".:.local/bin:/opt/homebrew/sbin:/opt/homebrew/bin:$PATH:$HOME/bin"
export vpc_id="vpc-14814b73"
export subnet_id="subnet-f02c90ab"
export ami_id="ami-014d05e6b24240371"
export keypair_name="rebecca_app"
```

```
export proj_name="rebecca_app"
export instance_type="t2.micro"
export storage_size="30"
export ami_id="ami-014d05e6b24240371"
export security_grp="sg-008cace70d32f6267"
export static_ip='13.56.101.209'

if type rg &> /dev/null; then
  export FZF_DEFAULT_COMMAND='rg --files --hidden --no-ignore-vcs'
  export FZF_DEFAULT_OPTS='-m --height 50% --border'
fi

export ZSH_AUTOSUGGEST_HIGHLIGHT_STYLE="fg=011,bg=black,bold,underline"
LS_COLORS+='pi=01;33:so=01;33:do=01;33:bd=01;33:cd=01;33:su=01;35:sg=01;35:ca=01;35:ex=01;32
export LSCOLORS='ExGxDxDxCxDxDxFxFxexEx'
```

4.3 ~/.config/zsh/.zsh_functions

```
function d () {
  if [[ -n $1 ]]; then
    dirs "$@"
  else
    dirs -v | head -n 10
  fi
}

# Mathematica script execution
mma () {
  /Applications/Mathematica.app/Contents/MacOS/WolframKernel -script $1
}

# Streamlined git workflow for data science projects
function gz() {
    git add .
    git commit -a -m "$1"
    git push
}

# Additional data science utility functions
function conda_env() {
```

```
    conda activate "$1" && echo "Activated environment: $1"
}

function jupyter_start() {
    jupyter lab --no-browser --port=${1:-8888}
}

function r_install() {
    Rscript -e "install.packages('$1', repos='https://cran.rstudio.com/')"
}
```

4.4 Function Categories

- **Directory Management**: Efficient navigation and workspace organization
- **Version Control**: Streamlined git operations for iterative data analysis
- **Environment Management**: Quick activation of conda/virtual environments
- **Tool Integration**: Seamless launching of Jupyter, R, and computational tools

5 Results and Key Findings

Implementing this terminal configuration provides significant productivity improvements for data science workflows:

1. **Enhanced Navigation Efficiency**: Directory jumping and smart aliases reduce navigation time by ~60%
2. **Improved Visual Feedback**: Syntax highlighting and custom prompts minimize command errors
3. **Streamlined Tool Access**: Custom functions and aliases reduce repetitive typing for common tasks
4. **Better Workspace Organization**: Modular configuration files enable easy customization and maintenance

Figure 2: UCSD Campus Library - Modern workspace for academic and research excellence

The collaborative research environment at UCSD exemplifies the kind of productive workspace that a well-configured terminal can help create for data science development

6 Limitations and Considerations

While this configuration significantly improves terminal productivity, there are some important considerations:

6.1 System Dependencies

- **Package managers required**: Homebrew (macOS) or equivalent Linux package managers
- **Font requirements**: FiraCode Nerd Font needed for optimal visual experience
- **Plugin compatibility**: Some plugins may require specific versions of Zsh

6.2 Performance Considerations

- **Startup time**: Multiple plugin loading can increase shell initialization time
- **Memory usage**: Enhanced features consume additional system resources
- **Compatibility**: Configuration may need adjustment across different operating systems

6.3 Customization Limitations

- **Personal preferences**: Aliases and shortcuts reflect individual workflow patterns
- **Project-specific needs**: Some configurations may not suit all data science project types
- **Team collaboration**: Highly customized environments may not transfer to shared systems

7 Future Extensions

This configuration could be enhanced in several directions:

- Integration with containerized development environments (Docker, Podman)
- Advanced git hooks for automated data validation and testing
- Custom plugins for specific data science frameworks (TensorFlow, PyTorch)
- Integration with cloud development environments and remote computing resources
- Automated backup and synchronization of configuration across multiple machines

8 Conclusion

A well-configured terminal environment is essential for efficient data science development. This setup provides a solid foundation with Kitty terminal and Zsh shell, enhanced by productivity plugins, custom aliases, and organized configuration files.

Key Benefits: - Faster navigation and file management - Reduced typing through intelligent aliases and functions - Better visual feedback and error prevention - Modular, maintainable configuration structure

Next Steps: - Implement the configuration on your development machine - Customize aliases and functions for your specific workflow - Explore additional plugins and tools mentioned in the references

I encourage you to adapt these configurations to your specific data science workflow and share your customizations with the community.

9 References and Further Reading

9.1 Technical Documentation

1. **Terminal and Shell Configuration:**
   - Kitty Terminal Documentation - Comprehensive configuration guide
   - Zsh Documentation - Official Zsh manual and scripting guide
   - Oh My Zsh Framework - Popular Zsh configuration framework
2. **Productivity Tools:**
   - FZF Documentation - Command-line fuzzy finder
   - Ripgrep User Guide - Fast text search tool
   - Eza Documentation - Modern replacement for ls command

9.2 Blog Posts and Tutorials

1. **Terminal Setup Guides:**
   - Configuring Zsh Without Dependencies - Minimal Zsh configuration approach
   - My Terminal Setup: iTerm2 + ZSH + Powerlevel10k - Comprehensive terminal customization
   - Settings For a Better iTerm2 Experience - Performance and usability optimization
2. **Development Workflow:**
   - iTerm2 Setup and Customization - Detailed configuration walkthrough
   - Command Line Tools for Data Science - Comprehensive CLI data science guide
   - Terminal-Based Data Science Workflows - Integration with Jupyter and other tools
3. **Advanced Configuration:**
   - Dotfiles Management Best Practices - Version control for configuration files
   - Shell Scripting for Data Scientists - Automation and scripting techniques
   - Cross-Platform Terminal Configuration - Example configurations

9.3 Community Resources

1. **Forums and Discussion:**
   - r/commandline - Terminal tools and configuration discussions
   - Unix & Linux Stack Exchange - Technical troubleshooting and tips
   - Zsh Users Mailing List - Official community support
2. **Configuration Repositories:**

- Awesome Dotfiles - Curated list of configuration examples
- Dotfiles.org - Community-shared configurations
- GitHub Dotfiles - Version control best practices

10 Reproducibility Information

10.1 Environment Requirements

- **Operating System**: macOS 10.15+ or Linux (Ubuntu 18.04+, Fedora 30+)
- **Package Manager**: Homebrew (macOS) or system package manager (Linux)
- **Dependencies**: Git, Zsh 5.0+, basic development tools

10.2 Configuration Files

- **Repository**: Configuration files available in the post's code blocks
- **Installation**: Copy configurations to appropriate directories (~/.config/zsh/, ~/.zshrc)
- **Backup**: Always backup existing configurations before implementing changes

10.3 Version Information

```
# Check versions of key components
zsh --version          # Zsh version
kitty --version        # Kitty terminal version
brew --version         # Homebrew version (macOS)
```

10.4 Share This Post

Found this helpful? Share it with your network:

- Twitter
- LinkedIn
- Reddit

10.5 Connect and Discuss

Have questions about terminal configuration or suggestions for improvements?

- **Twitter**: @rgt47 - Quick questions and discussions
- **LinkedIn**: Ronald Glenn Thomas - Professional networking
- **GitHub**: rgt47 - Code, issues, and contributions
- **Email**: Contact through website - Detailed inquiries

10.6 About the Author

Ronald (Ryy) Glenn Thomas is a biostatistician and data scientist at UC San Diego, specializing in statistical computing, machine learning applications in healthcare, and reproducible research methods. He develops R packages and conducts research at the intersection of statistics, data science, and clinical research.

Connect: Website | ORCID | Google Scholar