
TERMINAL VELOCITY
elprztwi
!lurk
streamlabs ad3elprztwi is now lurking
irc: disconnected from server
rwxrob (rwxrob@rwxrob.tmi.twitch.tv) has joined #rw
Channel #rwxrob: 1 nick (0 ops, 0 voiced, 1 regular
s for 2025
rwxrob.github.io/blog
Thu Jan 2

# Terminal Velocity

Robert S. Muhlestein (rwxrob)

Version v0.0.1, 2025-01-02 21:26:24: Under development

# Table of Contents

*Boost your coding, hacking, and learning with the fastest human-computer interface*

| https://linktr.ee/rwxrob | Download PDF | Download EPUB |
|---|---|---|

# Copyright

# Dedication

**In memory of Aaron Swartz and Kris Nova**,
whose contributions and spirit continue to inspire the pursuit of knowledge and innovation.

# Preface

`Hello, friend.` Exciting time in the world. Exciting time. Systems are breaking. Power structures are crumbling. Climate fluctuating. The ones who control the code, control the narrative and the solutions. You see it, don't you? Technology is reshaping everything—how we live, work, and connect. Quantum and AI advancements are growing at exponential rates of exponential rates. The terminal isn't just a relic of the past; it's the key to survival at this pace, understanding and mastering the systems that drive the world forward. *Everything* can be accessed from a terminal.

> "What is the most important thing you could be working on in the world right now? And if you're not working on that, why aren't you?" (Aaron Swartz)

Those not moving at *terminal velocity* will be left behind. While most of the world flails in chaos, grasping for the next game, streaming video, or shiny desktop distraction, those who learn and use the terminal see through the noise, into the Matrix, where they automating processes, secure networks, and provide solutions to the world's most dire and complex problems. Like anything worth learning, these skills take commitment. Are you ready? Willing to put in the time? If so, welcome to the edge, knowledge warrior. Now let's get to work.

Data Science might be the fastest growing tech profession in 2024 (36% growth rate)—having recently bested Security Analyst (33%) for that position—but you need terminal skills for both.

I once watched a remote presentation from a data scientist who didn't know about `mv name newname`. 100 people watched him drop out of the terminal—where he was already showing everyone how to do some data-science things—just so he could rename a file. He minimized his terminal, opened the File Explorer, renamed the file using the GUI properties dialog box, then reopened the terminal to continue. I couldn't believe it. Still blows me away today. It's one of the reasons I wrote this. Don't get mad, get busy.

## Who should read this?

This book is for anyone over thirteen who wants to master the terminal like a hacker. If you're under thirteen? I won't know. But you should get your parents' permission—seriously. Services like these don't mess around when it comes to COPPA, and for good reason. If you're younger than thirteen, skip the parts of this book that require accounts or tools with age restrictions. Some schools provide access to these services for kids. Ask. Hack the system the right way.

## The hacker's way

In this book, *The Hard Way* is better known as *The Hacker's Way*. It's not about doing things the easy way or even the fast way—it's about doing them *your* way. This path teaches you to think like a hacker: to see every gap as an opportunity, every problem as a puzzle, and every limitation as a challenge to overcome.

Here, we focus on *what* to learn rather than spoon-feeding you the *how*. You'll find guidance on the tools, concepts, and techniques that matter, but the specifics? Those are yours to discover. The Hacker's Way demands that you go beyond the pages of this book, leveraging your curiosity, creativity, and the boundless resources of the digital world to piece it all together.

Why this approach? Because hacking isn't about following instructions; it's about breaking them down, understanding them, and reimagining them. The act of figuring out how something works on your own—not just by imitating but by innovating—is what transforms you into a true hacker. It builds not just technical skills but also resilience, independence, and an instinct for solving problems on your terms.

When this book leaves out the *how*, it's intentional. It's your cue to dig deeper. Fire up your search engine, dive into documentation, and collaborate with your AI assistants. Don't just find the answer—dissect it, experiment with it, and make it yours. The Hacker's Way is about mastering the process, not just the outcome.

This is a mindset for those who thrive on challenges and who are willing to embrace the unknown. It's for the autodidacts, the problem solvers, and the terminal knowledge warriors. If you choose The Hacker's Way, you'll gain something far more valuable than shortcuts: you'll gain the ability to create your own.

## Write your own book

For a hacker, the best book isn't something you buy—it's something you create. Following a textbook might teach you the basics, but it won't sharpen the skills that truly matter: curiosity, problem-solving, and adaptability. Hackers learn by doing, breaking things, and figuring out how to put them back together better. Writing your own book filled with original notes in Markdown that includes runnable code saved with Git is the ultimate extension of that process.

When you create your own book, you decide what's important. Instead of passively consuming information, you actively shape it, curating concepts and methods that matter to you. You remember it better. This approach forces you to think deeply, experiment, and learn at your own pace. Every chapter you write becomes a record of what you've discovered, how you've applied it, and what you've learned in the process.

By building your own resource, you're not just learning—you're documenting your journey in a way that's meaningful and personal. Unlike a static textbook, your book evolves with you, reflecting the growth of your knowledge and skills. It's not just a guide; it's a testament to your ability to hack the learning process itself.

## Wax on, wax off

Повторение — мать учения. Repetition is the mother of learning. Anyone who has ever done anything worth learning knows not all procedures can be memorized and ready. You have to come back to them and repeat them. This is why you setup your own learning lab where you take dynamic notes that change as the times do and as you learn more. It's the reason this book is perpetually published. It's never finished. A hacker's learning never is. Always adapting and acquiring new skills and knowledge. Once you think you've learned it all, you get taken out.

# Join a crew

We humans are social creatures. Learning is communal. Join a community that is focused on learning the same things as you. Meetups, hacker spaces, clubs, church groups, social media, and live streamer chat groups are all great places to find a community. The community is a place to collaborate and share peer reviews of one another's learning and projects. Who knows, you might find a friend. I did.

My sincere hope is that the parents, teachers, and young hackers reading these words will find opportunities and motivation to create their own clubs and communities dedicated to helping others master the art of the terminal command line and bash scripting and all the well-earned benefits of doing so. While you are working on creating your own community, consider joining my crew: https://linktr.ee/rwxrob.

# Find a mentor

Humans have been passing knowledge one to another since the beginning. It's what we do. Wanna forge a sword or horse shoe? Better find the blacksmith and get them to show you their craft. Coding is no different. Just don't ask them to *teach* you. They are probably too busy paying their bills doing what they are good at rather than spending that time teaching you how to do it. If you are respectful of their time, however, they might just be willing to mentor you—especially if you show genuine interest in their craft.

You will find that many people *want* to share what they know but just don't know how to do it or who would want to hear. Find these people and approach them. Perhaps you can find a few mentors so you can compare how they differ in their approach to programming.

# Become a mentor

Here's a secret a lot of people who get paid a lot of money don't want you to know, you don't need any special training to help someone else learn. In fact, as a beginner who has recently mastered something you are perhaps the best suited to help another beginner learn it because what made it click for you is still fresh in your mind. Ask a veteran when the last time anything clicked for them? They get so used to what they do they completely forget what it was like to be a beginner. So don't be shy. Offer your help to another beginner. And remember, you really haven't learned it until you can help someone else learn it as well.

# ChatGPT for learning

In my experience ChatGPT is an absolute necessity for anyone wanting to truly take their learning to the next level. I'm not getting paid to say that (but I would happily accept money for saying so). You really should consider subscribing if you are serious about adopting an autodidactic lifestyle. ChatGPT is specifically designed for general learning and creativity and runs on anything with full contextual history. Mine has helped me keep my conversational Russian and French skills up, planned bike trips, helped me code the mundane stuff, reminded me of things I forgot about,

offered up random ideas, and so much more.

Claude, another AI assistant, is better for `mods` terminal API integration and for coding.

Still skeptical? I know I was.

Here's the thing. On demand learning exponentially increases when an AI is involved. Nothing breaks through frustration and loneliness better when taking on learning challenges with a supportive AI companion—even when a helpful human mentor is also available.

We are quickly approaching a time when the digital divide will no longer be just between those who have computers and Internet access and those who do not, but between those who have learned to leverage a personal AI assistant loaded with contextual history and those who have not. Don't get left behind.

We are already seeing this difference around us every day. I'm remarkably faster at the same job now with AI doing the same thing I've been doing for four years, and that's not even just the coding part. Querying an AI is exponentially better for research than a Google search, provided I verify the results, as always.

# Trust me, kiddo

Most of us have to commit to one school (and no, not the kind with lockers and hall passes) to get started. Like Neo with Morpheus, there's got to be some level of trust—enough to take that first leap. But here's the kicker: you shouldn't trust anyone, ever. Not until they've earned it. And yet, how does someone earn your trust without you taking the risk to trust them first? It's a paradox, a mind-bending loop that feels impossible to solve. But it's in that uncertainty, in that leap of faith, where the real learning begins.

Commitment to a single school does not mean there is no value in others, or even that this school is better. Just that you are dedicated to this one at the moment. You would never walk into a Karate dojo and start talking about why Judo is a superior martial art. The masters of both dojos can even strongly respect the other for their commitment despite their different approaches. The master of the dojo need not justify why their form is better. That is up to the student to determine on their own after they have mastered the forms. A master and mentor, instead, focus uniquely on the advantages of the form they know and how to execute it.

Learning the terminal follows the same philosophy. At first, you're typing commands, opening multiple concurrent windows, and running programs without fully understanding how it all connects. By focusing on the terminal environment and immersing yourself in bash's syntax and tools, you're laying a foundation that will make advanced concepts easier to grasp later.

By dedicating yourself to the terminal you're committing to a single, cohesive system. The command line's simplicity and clarity make it an excellent first experience with programming. Every command line *is* a line of code. The terminal, as your dojo, reinforces this focus by stripping away distractions and emphasizing direct, hands-on practice.

Through this focused practice, you begin to see the "why" behind the techniques, and the lessons

become part of your intuition, your faith transforms into fact. So dedicate yourself to *this* practice right now.

Once you have mastered the basics of a single discipline, you are prepared to branch out. A martial artist with a strong foundation in one style can explore others and integrate their techniques into a unique, personal expression. Similarly, a terminal hacker with a host of commands at their fingertips can easily add new ones—even create their own—by applying the foundational skills they've gained.

# Dear parents and pedagogues

Are you a parent or a teacher? If so, thank you for being *awesome*! It takes a lot to bring kids into this world and even more to dedicate your life to helping kids that aren't even your own to find their way and learn something even if they don't want to. Deep respect. But let's get real. You're reading this right now wondering if I'm going to corrupt your children.

The answer is yes. Yes I am. In fact, my goal is to fully corrupt them much the same as Socrates. If I am successful your children will become *better* that the status quo, they'll think critically, ask uncomfortable questions, and use their new powers and tools to ethically disrupt this broken world transforming it into something better. They'll become the most annoying people you've ever known—in the best way possible.

Why me? Because getting people to discover this potential and realize it through their own learning is my super power. I've been mentored by the best and I'm *really* good at it. Since the first time my scouts begged "Mr. Rob, teach us to code. Mr. Rob, teach us to hack. Mr. Rob, teach us Russian." I have been bound and determined to make this a reality. In fact, I started SKILSTAK Coding Arts in 2013 with my own retirement money to address the *real* needs of the tech industry and those who might consider it as a career, raising a generation of highly skilled knowledge warriors.

Let me tell you about some of these amazing people:

- One of them quit his grocery clerk job to write code for a solar energy company and was paired with essentially his own intern at 16.
- Another couldn't get a job with a psychology degree so he learned this stuff and got a starting salary over $100,000.
- A few people got coveted invitations to special FBI cybersecurity camps.
- One drove 40 minutes both ways just to attend our sessions for *four years* and went on to create an entire coding club at her rural school.
- Some got cybersecurity degrees from accredited colleges and invites to work with professional hacking crews.

It's been a blast being with these people. My favorite memories are all the times we hacked companies with permission and helped them patch their security, or shut down hackers for panicked parents whose businesses had been compromised.

The point is, these success stories don't have to be about someone else. They can be about someone *you know and love*. Terminal tech skills really do change lives and I'm here to help in whatever way

I can.

# Your script

> The code *is* the documentation.

What better way to learn to code than programming yourself with *actual* code. Here you will find everything you need to do and learn to optimize your personal performance with a Unix-like terminal workflow in bash, a real language.

Not only will you be learning to use the terminal, you'll become an interpreter as you read each line, evaluate it, perform what it says, and loop to repeat that process for a new line, which is a real thing called a REPL. In a very real sense, you take the place of the bash command shell interpreter running on your computer. After all, the only difference between you and bash is your squishy gray persistence layer.

Now would be a good time to go watch that scene from Star Wars, A New Hope, where C3PO introduces himself as an *interpreter* who converts languages into instructions other devices understand, like "Bocce" and "the binary language of moisture evaporators." You'll have a lot in common with C3PO while working through this script.

## What *is* a script?

An actor's script and a bash script, though from vastly different domains, share a striking similarity: both are carefully crafted sets of instructions designed to guide actions and interactions. In an actor's script, the lines of dialogue and stage directions dictate the performance, instructing the actor on what to say, how to say it, and how to move or react. These instructions form a blueprint for storytelling, where the actor becomes the interpreter, turning static words into dynamic expressions.

Similarly, a bash script provides a series of commands that a computer interprets and executes, automating tasks or orchestrating processes. Just as an actor's script might include cues for a dramatic pause or a sudden outburst, a bash script might contain conditional statements or loops, signaling the system to pause, evaluate conditions, or repeat actions. Both types of scripts rely on precise language to avoid misinterpretation—whether by the actor or the computer—and both have an audience: the viewer for the actor's script, and the user or developer for the bash script.

Ultimately, both scripts transform abstract instructions into meaningful action. An actor brings a script to life by adding emotion and timing, while a bash script comes alive when executed, interacting with files, systems, or other programs. In both cases, the effectiveness of the script depends on the clarity of its instructions and the skill of its interpreter.

## Execute this

In coding, the term *execute* simply means to carry out or perform a set of instructions. When you execute a script or a program, you're telling the computer to follow the sequence of commands it contains and perform the tasks specified. This usage comes from the idea of execution as "putting

into effect" or "bringing something to life." It's entirely unrelated to the darker connotation of the word associated with capital punishment.

In programming, "execution" is a vital concept—it transforms static lines of code into dynamic processes, turning logic and algorithms into actions that produce results. Whether it's running a script to clean up files, compiling code into an application, or querying a database, execution is what bridges the gap between planning and doing. For learners, it's helpful to remember that when we say "execute," we're referring to the moment a computer takes our written instructions and makes them real.

Time for you to execute. Here's your script. Now execute it. Follow each line and when you come to a new one, see if there is a *subroutine* for that line that has more lines. You can think of it as a textual flow-chart, because that is *exactly* what it is. You got this. Let's go.

```
#!/bin/sh
set -e

# TODO: finish this, still a work in progress

use_unix_terminal() {
    check_prerequisites
    #setup_terminal
    #navigate_file_system
    #manipulate_files
    #edit_files_with_text_editor
    #manage_screens_with_tmux
    #code_simple_shell_scripts
    #manage_source_with_git
    #manage_development_with_github
    #configure_interactive_shell
    #configure_vim_nvim
    #use_web_from_terminal
    #use_ai_from_terminal
    #use_irc_from_terminal
    #manipulate_text_with_perl
    #plan_next_move
}

check_prerequisites() {
    have_basic_linear_algebra # y = 2x + 3
    #have_6th_grade_reading_level  # ex: The Boy Who Harnessed the Wind
    #have_compatible_computer      # intel, apple, raspi, orangepi
    #have_admin_rights_on_computer # root, administator
    #have_internet_access          # basic cable connection
}

have_basic_linear_algebra() {
    assert \
        "Do you have basic linear algebra skills?" \
        "Consider Kahn Academy."
```

```
}

# ---------------------------- utilities ----------------------------

assert() {
    printf "%s [N|y] " "$1"
    read -r resp
    if [ "$resp" != y ]; then
        echo "$2"
        exit
    fi
}

use_unix_terminal

# ---------------------------- old stuff ----------------------------

: <<-'EOM'
    setup_terminal_on_workstation() {
        :start_using_computer_efficiently # launcher, alt_tab, search_centric
        :start_using_unix_like_system     # Ubuntu Linux, macOS, WSL2
        :start_using_package_managers     # apt, brew, winget.exe

        read _rp "Do you care about colors and fastest terminal? [Y|n]" resp
        if [[ ${resp,,} =~ ^y ]]; then
            :start_using_wezterm_terminal
        else
            :start_using_builtin_terminal
        fi
    }

    # Note that this includes both default PowerShell on Windows
    # as well as bash, zsh, and any other POSIX Unix shell.
    start_using_basic_shell_commands() {
        :start_working_with_files_from_command_line
    }

    start_working_with_files_from_command_line() {
        local mode=command_line_only
        :grok_file_system_heirarchy
        :grok_drive_mounting
        :grok_how_everything_is_a_file_in_unix
        :start_listing_files
        :start_navigating_file_system
        :start_finding_files_with_find_command
        :start_working_with_file_permissions
        :start_seeing_whats_in_files
        :start_modifying_files_without_editor
        :start_organizing_files
        :start_linking_files
    }
```

```
    start_using_web_from_terminal() {
        start_fetching_single_web_pages_with_curl
        start_browsing_web_with_w3m
        fetch_initial_lynx_config
        start_browsing_web_with_lynx # requires bash scripting skill
    }

    start_connecting_with_other_terminal_lovers() {
        start_watching_terminal_geeks_on_youtube
        start_chatting_with_twitch_terminal_streamers
        start_using_weechat_for_twitch_and_irc
        start_chatting_on_preferred_irc_channels
    }

    start_creating_advanced_terminal_apps() {
        :grok_map_filter_reduce_transform
        :start_creating_advanced_bash_commands  # parameter expansion, hashes, etc.
        :start_creating_terminal_commands_in_go # flags, bonzai, cobra, bubbletea
        :learn_enough_lua_for_dynamic_configs   # ~/.wezterm.lua,
~/.config/nvim/init.lua
    }

    start_managing_stuff_with_git() {
        grok_git_essentials        # just for local backups
        start_using_bare_git_repos # just on local computer
        start_using_github         # but not necessarily for everything
        start_dot_repo             # often called "dotfiles"
        start_lab_repo             # for notes and testing
    }

    start_creating_stuff_using_terminal() {
        start_editing_text_from_terminal  # nano → ed → ex → vi (nvi) → vim → nvim
        start_using_terminal_multiplexers # screen → tmux
        start_taking_notes_in_markdown    #specifically GitHub Flavored Markdown
        start_writing_basic_bash_scripts  # but not too many until git
        start_custom_bashrc               # more than just ~/.bashrc
    }

    start_creating_advanced_bash_commands() {
        create_filters_in_bash
    }

    declare _i TRUE=0
    declare _i FALSE=1
    #declare TERMPREF=wezterm

    start_using_dotfiles_repo() {
        : TODO pull in tmux.conf
    }
```

```
start_writing_scripts_in_bash() {
    : grok_posix_vs_bash
    :customize_bashrc
}

start_using_with_irc() {
    :
}

start_using_unix_like_system() {
    start_using_basic_universal_shell_commands
    grok_history_of_unix
    grok_why_unix
    get_unix_like_system
}

start_using_wezterm() {
    launch_default_terminal
    install_wezterm
    config_wezterm_with_nano
}

start_using_package_managers() {
    grok_package_managers
    setup_package_manager
}

start_using_terminal_multiplexers() {
    grok_multiplexers
    install_tmux
    config_tmux_like_screen
    learn_tmux
}

config_tmux_like_screen() {
    : TODO give option to pull down reliable tmux.conf from Web
}

grok_multiplexers() {
    grok_screen_history
    grok_tmux_history
    if agree_to_config_tmux_like_screen; then
        download_tmux_screen_config
        apply_tmux_screen_config
    fi
}

download_tmux_screen_config() {
    :
}
```

```
apply_tmux_screen_config() {
    :
}

agree_to_config_tmux_like_screen() {
    :
}

grok_screen_history() {
    :
}

grok_tmux_history() {
    :
}

start_editing_text_from_terminal() {
    : customize_vim
    : customize_neovim
}

start_writing_your_own_docs() {
    :
}

start_using_git_and_github() {
    :
}

validate_computer_specs() {
    :
}

learn_alt_tab() {
    :
}

learn_launcher() {
    :
}

prefer_search_centric_navigation() {
    :
}

grok_why_unix() {
    :
}

grok_package_managers() {
    :
```

```
}

setup_package_manager() {
    :
}

launch_default_terminal() {
    :
}

install_wezterm() {
    :
}

start_using_computer_efficiently() {
    learn_alt_tab
    learn_launcher
    prefer_search_centric_navigation
}

config_wezterm_with_nano() {
    :
}

grok_unix_history() {
    :
}

launch_builtin_terminal() {
    if is_mac; then
        launch "terminal"
    elif is_win; then
        launch "wsl.exe" # we assume WSL2 was already installed
    elif is_lin; then
        : # TODO
    fi
}

press() {
    printf "Press the %s key." "$1"
}

enter() {
    printf "Type in '%s' and press Enter." "$1"
}

launch() {
    if is_mac; then
        :
    elif is_win; then
        press 'Win'
```

```
            enter "$1"
        elif is_linux; then # FIXME which linux
            :
        fi
    }

    have_mac() {
        if [[ "$OSTYPE" =~ ^darwin ]]; then
            return $TRUE
        else
            return $FALSE
        fi
    }

    get_unix_like_system() {
        if have_mac; then
            install_brew
        fi
    }

    install_brew() {
        echo would install brew
    }

    boost_to_terminal_velocity
EOM
```

# Terminology

Let's cover some terms that will help understand what you have been doing. You don't have to read them all now, but see if you can identify them in the code that follows:

- Procedure - a set of instructions to accomplish a *task*
- Scope - how high or low the level of instructional detail
- Algorithm - mostly just a fancy way to say *procedure*
- Task -
- Skill -
- Ability -
- Knowledge -
- Operation -
- Parameter - a variable passed into a *function*, *method*, or *procedure* from an *argument*
- Argument - a value assigned to a *parameter* variable
- Variable -
- Constant -
- Type -
- Object -
- Method -
- Function - optionally put something in, get something out, like a machine
- Subroutine - a reusable block of code including methods, functions, procedures