Part 0 - Terminal Survival Guide

A quick reference for when things go wrong

The Super Basics

Where Am I and What's Here?

Finding Your Location

```
pwd                     # Print Working Directory - shows where you are
```

Listing Files

```
ls                      # List files in current directory
ls -l                   # Detailed list (permissions, size, date)
ls -la                  # Include hidden files (start with .)
ls -lh                  # Human-readable sizes (KB, MB, GB)
ls *.txt                # Only .txt files
```

Understanding Paths

- Absolute path: `/home/username/Documents` (full path from root)
- Relative path: `Documents/project` (relative to current location)
- `~` = your home directory
- `.` = current directory
- `..` = parent directory (one level up)
- `/` = root directory (Linux/Mac) or drive letter on Windows

Moving Around

```
cd Documents            # Go into Documents folder
cd ..                   # Go up one level
cd ../Downloads         # Go up, then into Downloads
cd ~                    # Go to home directory
cd                      # Also goes to home
cd -                    # Go back to previous directory
```

Tab Completion - USE THIS!

- Start typing a filename/command and press Tab
- It will autocomplete or show options
- Saves time and prevents typos
- Press Tab twice to see all possibilities

File Operations

Creating and Managing Files

Basic File Operations

```
touch file.txt        # Create empty file or update timestamp
cp source dest        # Copy file
cp -r folder/ dest/   # Copy directory
mv old new            # Move or rename
rm file               # Delete file
rm -r folder/         # Delete directory
mkdir folder          # Create directory
rmdir folder          # Delete empty directory
```

Shortcuts

```
# Create and enter directory
mkdir folder && cd folder

# Make multiple directories at once
mkdir -p project/{src,tests,docs}
```

Viewing and Reading Files

Viewing Files

```
cat file.txt          # Show entire file (small files only!)
head file.txt         # First 10 lines
head -20 file.txt     # First 20 lines
tail file.txt         # Last 10 lines
```

Safe Paging

```
less file.txt         # View large files safely
```

Inside `less`

- `Space` - Page down
- `b` - Page back up
- `/word` - Search for “word”
- `n` - Next search result
- `N` - Previous search result
- `q` - Quit

Never use `cat` on huge files! Use `less` instead.

Windows Equivalents

```
cat file.txt           # Works in PowerShell
cat file.txt | more    # `more` works like `less`
```

Text Processing Basics

```
grep pattern file     # Search for pattern
grep -i pattern file  # Case-insensitive
sort file             # Sort lines
uniq file             # Remove duplicate lines
cut -d',' -f1 file    # Extract first CSV column
```

Counting Things

```
wc -l file.txt        # Count lines
wc -w file.txt        # Count words
wc -c file.txt        # Count bytes/characters
ls | wc -l            # Count files in directory
```

Downloading Files

```
# Saves as quest0.tar.gz in the current directory
wget https://terminal.atlasdev.club/quest0.tar.gz

# Does the same. This works on Windows too
curl -fSSL https://terminal.atlasdev.club/quest0.tar.gz -o quest0.tar.gz
```

Finding Files and Commands

Searching for Files

By Name

```
find . -name "*.py"             # All Python files
find . -name "*report*"         # Files with "report" in name
find . -type f -mtime -7        # Files modified in last 7 days
find . -type d -name "test*"    # Directories starting with "test"
find . -size +100M              # Files larger than 100MB
```

By Content

```
grep -r "TODO" .                # Search all files for "TODO"
grep -r "import pandas" .       # Find files importing pandas
grep -rn "error" logs/          # Search with line numbers
```

Windows Equivalents

```
dir /s *.py           # Find Python files recursively
findstr /s "TODO" *   # Search file contents
```

Finding Commands

Command Location

```
which python          # Where is python?
whereis python        # Find binary, source, manual
type cd               # What kind of command is cd?
```

Windows Equivalents

```
where python          # Find python location
Get-Command python    # PowerShell command info
```

Command History - Your Best Friend

Searching History

```
Ctrl+R                # Reverse search - start typing to find
                      # Press Ctrl+R again to cycle through matches
                      # Press Enter to run, Right Arrow to edit

history               # Show all past commands
history | grep git    # Find git commands you've run
history | sls git     # Above, on Windows
history | tail -20    # Last 20 commands
```

Reusing Commands

You can press the Up and Down Arrow keys to go through your previous commands in order. The commands below don't work on Windows.

```
!!                    # Repeat last command
sudo !!               # Run last command with sudo
!$                    # Last argument of previous command
!123                  # Run command number 123 from history
```

Example Usage

```
# Forgot to use sudo?
apt install package   # Permission denied
sudo !!               # Runs: sudo apt install package

# Reuse last argument
mkdir /long/path/to/folder
cd !$                 # Goes to /long/path/to/folder
```

Working with Archives

Extracting Files

```
tar -xzf file.tar.gz  # Extract .tar.gz ("eXtract Ze File")
tar -xjf file.tar.bz2 # Extract .tar.bz2
tar -xf file.tar      # Auto-detect format
unzip file.zip        # Extract .zip
```

Creating Archives

```
tar -czf backup.tar.gz folder/    # Create .tar.gz
zip -r archive.zip folder/        # Create .zip
```

Viewing Archive Contents

```
tar -tzf file.tar.gz  # List contents without extracting
unzip -l file.zip     # List zip contents
```

Windows Equivalents

`tar` works as usual on Windows. For zip commands,

```
Expand-Archive file.zip
Compress-Archive folder/ archive.zip
```

Permissions and Access

Understanding File Permissions

Reading `ls -l` Output

```
-rw-r--r--  1 user group 1234 Jan 1 12:00 file.txt
|||||||||
|└└└└└└└└─ Permissions
└────────── File type (- = file, d = directory, l = link)
```

```
rwx rwx rwx
|   |   |
|   |   └── Others (everyone else)
|   └────── Group (users in file's group)
└────────── Owner (file owner)
```

```
r = read (4)
w = write (2)
x = execute (1)
```

Common Permission Numbers

- `755` = rwxr-xr-x (owner: all, others: read+execute)
- `644` = rw-r--r-- (owner: read+write, others: read)
- `600` = rw------- (owner: read+write, others: nothing)
- `777` = rwxrwxrwx (everyone: everything - usually bad!)

Fixing Permission Problems

Making Scripts Executable

```
./script.sh           # Permission denied
chmod +x script.sh    # Make it executable
./script.sh           # Now it works!
```

Common Permission Commands

```
chmod +x file         # Add execute permission
chmod 755 script.sh   # Set specific permissions
```

```
chmod -R 755 folder/  # Change folder and contents recursively
chown user:group file # Change owner (may need sudo)
```

Using Sudo

```
sudo command          # Run as administrator/root
sudo !!               # Run previous command with sudo
su                    # Become root user (be careful!)
```

Windows Equivalents

- If you have it, enable sudo from developer settings
- Right-click → “Run as Administrator”
- No direct chmod equivalent, but you probably don’t need it

Running and Managing Processes

Starting Programs

Running Programs and Scripts

```
python script.py      # Run a Python script
./script.sh           # Run a shell script in current directory
./program             # Run an executable in current directory
```

The `./` means “in the current directory”

Process Control

Stopping and Pausing

```
Ctrl+C                # Stop current command (most common!)
Ctrl+Z                # Suspend/pause current command
Ctrl+D                # End input / exit shell
```

Managing Background Jobs (only on Linux/WSL)

```
jobs                  # List suspended/background processes
fg                    # Bring suspended job to foreground
fg %[id]              # Bring job ID [id] to foreground
bg                    # Resume suspended job in background
```

Finding Processes

```
ps aux | grep python  # Find Python processes
top                   # Live view of all processes (q to quit)
htop                  # Better version of top (if available)
```

Killing Processes

```
kill 1234             # Kill process with ID 1234 (graceful)
kill -9 1234          # Force kill (last resort)
pkill firefox         # Kill by name
```

Windows Equivalents

```
# In PowerShell:
Get-Process firefox   # Find processes by name (like pgrep)
Get-Process *fire*    # Partial name match
tasklist              # List processes (like ps)
taskkill /PID 1234    # Kill by process ID
taskkill /IM name.exe # Kill by name
```

System Diagnostics

Checking Performance

CPU and Memory

```
top                   # Live process monitor (q to quit)
htop                  # Better version (if available)
free -h               # Memory usage (Linux)
uptime                # System load and uptime
```

Windows Equivalents

```
taskmgr               # Task Manager
Get-Process           # PowerShell process list
```

Network Debugging

Testing Connectivity

Basic Tests

```
ping google.com       # Is the internet working?
ping 8.8.8.8          # Test by IP (bypasses DNS)
ping -c 4 google.com  # Send only 4 packets (Linux)

curl -I example.com   # Test HTTP connection
curl wttr.in          # Fun: get weather!
curl ifconfig.me      # Get your public IP
```

Port Testing

```
nc -zv host 80        # Check if port 80 is open
nc -zv localhost 3000 # Is my app listening on port 3000?
```

Windows Equivalents

```
ping google.com       # Same as Linux
Test-NetConnection -ComputerName google.com -Port 80
curl.exe example.com      # Works in PowerShell too
```

Port Conflicts

Finding What's Using a Port

```
lsof -i :8080                # What's on port 8080? (Mac/Linux)
netstat -tulpn               # List all used ports
netstat -tulpn | grep 8080   # Find what 8080 is being used by
ss -tulpn | grep 8080        # Modern alternative (Linux)
```

The "Address Already in Use" Problem

```
# Error: Address already in use
# Find the culprit:
lsof -i :3000         # Shows PID using port 3000
kill 1234             # Kill that process
# Now you can start your server
```

Windows Equivalents

```
netstat -ano | findstr :8080  # Find process on port
taskkill /PID 1234 /F         # Kill that process
```

Environment and PATH

Understanding Environment Variables

Viewing Variables

```
echo $HOME            # Your home directory
echo $PATH            # Where shell looks for commands
echo $USER            # Your username
env                   # Show all environment variables
env | grep PYTHON     # Find Python-related variables
```

Setting Variables

```
export VAR=value      # Set for current session
export PATH=$PATH:/new/path    # Add to PATH
```

Common "Command Not Found" Fix

```
# Installed something but "command not found"?
which command-name    # Is it installed?
echo $PATH            # Is the location in PATH?
export PATH=$PATH:/path/to/commands  # Add it
```

Windows Equivalents

```
echo $env:PATH          # View PATH
$env:VAR = "value"      # Set variable
[Environment]::SetEnvironmentVariable("VAR", "value", "User")
```

Reading Documentation

Quick Help

Command-Line Help

```
command --help          # Quick help for most commands
command -h              # Short version
```

Windows Equivalents

```
command /?              # Help in Command Prompt
Get-Help command        # Help in PowerShell
```

Online Resources

- explainshell.com - Paste a command, get explanation
- cheat.sh - Quick cheat sheets (`curl cheat.sh/tar`)

Man Pages (Built-in Manuals)

Linux/WSL only

Opening and Navigating

```
man ls                  # Open manual for ls command
man grep                # Manual for grep
man -k search           # Search all manuals for keyword
```

Inside a Man Page — same as `less`

- `Space` - Page down
- `b` - Page back up
- `/word` - Search for “word”
- `n` - Next search result
- `N` - Previous search result
- `q` - Quit

Understanding the Structure

- NAME: What the command is
- SYNOPSIS: How to use it (syntax)
- DESCRIPTION: Detailed explanation
- OPTIONS: All available flags
- EXAMPLES: Usage examples (if provided)

Terminal Disasters and Troubleshooting

Frozen Terminal

```
Ctrl+S                  # Accidentally freezes terminal
Ctrl+Q                  # Unfreezes it
```

b

Stuck in an Editor

```
# In vim:
:q!                     # Quit without saving
:wq                     # Save and quit

# In nano:
Ctrl+X                  # Exit (it shows hints at bottom)
```

Last Resort Commands

Terminal Completely Frozen

1. Try `Ctrl+C`
2. Try `Ctrl+Z` then `kill %1`
3. Try `Ctrl+Q` (if accidentally frozen with Ctrl+S)
4. Close the terminal window and open a new one
5. In another terminal: find and kill the process

Can't Edit Config Files

```
# Use nano instead of vim if stuck
nano ~/.bashrc

# Or reset to defaults
mv ~/.bashrc ~/.bashrc.backup
cp /etc/skel/.bashrc ~/
```

Broke Your Shell Config

```
# Most shells will still work with --norc
bash --norc
zsh --no-rcs

# Then fix the config file
nano ~/.bashrc
```

Quick Command Reference

File Operations

```
cp source dest          # Copy file
cp -r folder/ dest/     # Copy directory
```

```
mv old new              # Move or rename
rm file                 # Delete file
rm -r folder/           # Delete directory
mkdir folder            # Create directory
rmdir folder            # Delete empty directory
touch file              # Create empty file or update timestamp
```

Text Processing

```
grep pattern file       # Search for pattern
grep -r pattern .       # Recursive search
grep -i pattern file    # Case-insensitive
sort file               # Sort lines
uniq file               # Remove duplicate lines
cut -d',' -f1 file      # Extract first CSV column
```

Redirection and Pipes

```
command > file          # Redirect output to file (overwrite)
command >> file         # Append output to file
command 2> file         # Redirect errors to file
command1 | command2     # Pipe: output of cmd1 → input of cmd2
command < file          # Use file as input
```

Common Patterns

```
# Find and count
ls | wc -l              # Count files

# Search and save
grep error log.txt > errors.txt

# Chain commands
cat file | grep pattern | sort | uniq

# Multiple commands
command1 && command2    # Run cmd2 only if cmd1 succeeds
command1 || command2    # Run cmd2 only if cmd1 fails
command1 ; command2     # Run both regardless
```

Tips and Tricks

Keyboard Shortcuts

```
Ctrl+C                  # Stop current command
Ctrl+D                  # Exit shell / end input
Ctrl+L                  # Clear screen (same as 'clear')
Ctrl+A                  # Go to start of line
Ctrl+E                  # Go to end of line
Ctrl+U                  # Delete from cursor to start
Ctrl+K                  # Delete from cursor to end
Ctrl+W                  # Delete word before cursor
```

```
Ctrl+R                  # Search command history
Ctrl+Z                  # Suspend current process
```

Wildcards and Patterns

```
*                       # Matches anything
?                       # Matches single character
[abc]                   # Matches a, b, or c
[0-9]                   # Matches any digit
*.txt                   # All .txt files
file?.txt               # file1.txt, fileA.txt, etc.
```

Quick Wins

```
# Run previous command with sudo
sudo !!

# Create and enter directory
mkdir folder && cd folder

# Make multiple directories at once
mkdir -p project/{src,tests,docs}

# See folder sizes
du -sh */

# Find your largest files
du -ah . | sort -rh | head -20

# Save output of complex command
command | tee output.txt  # Shows AND saves
```

Remember

- **Silence is success** - no output often means it worked
- **Tab completion is your friend** - use it constantly
- **Ctrl+C stops most things** - don't be afraid to use it
- **Read the errors** - they usually tell you what's wrong
- **When stuck, try `--help` or `man`** - help is built in

Part 1 - Terminal Productivity Guide

Level up your command-line workflow

Productivity Multipliers

What Are Aliases?

- Shortcuts for common commands
- Simple text replacement, no parameters
- Perfect for frequently used commands

Basic Examples

```
alias ll='ls -lah'           # Detailed list view
alias ..='cd ..'             # Go up one directory
alias ...='cd ../..'         # Go up two directories
alias g='git'                # Quick git access
alias grep='grep --color'    # Colorize grep output
alias mkdir='mkdir -p'       # Create parent dirs automatically
```

Making Aliases Permanent

```
# Add to ~/.bashrc (Linux/WSL)
nano ~/.bashrc
# or ~/.zshrc (Mac with zsh)
nano ~/.zshrc

# After editing, reload:
source ~/.bashrc             # or source ~/.zshrc
```

Common Productivity Aliases

```
alias c='clear'
alias h='history'
alias ports='netstat -tulanp'
alias update='sudo apt update && sudo apt upgrade'
alias py='python3'
```

Parameterized Functions

When to Use Functions vs Aliases

- Aliases: Simple shortcuts, no logic needed
- Functions: Need parameters, conditions, or multiple commands

Useful Functions

```
# Create and enter directory
mkcd() {
    mkdir -p "$1" && cd "$1"
}
# Usage: mkcd new-project

# Find and edit files
fe() {
    local file
    file=$(find . -type f | fzf) && nano "$file"
}
# Usage: fe (then select file interactively)

# Git commit with message
gcm() {
```

```
    git add . && git commit -m "$1"
}
# Usage: gcm "fix: typo in readme"

# Create backup of file
backup() {
    cp "$1" "$1.backup.$(date +%Y%m%d_%H%M%S)"
}
# Usage: backup important.txt
```

Adding Functions Permanently

```
# Same as aliases - add to ~/.bashrc or ~/.zshrc
nano ~/.bashrc

# Then reload
source ~/.bashrc
```

Smart Navigation Tools

fzf (Fuzzy Finder)

Installation

```
# Linux/WSL
git clone --depth 1 https://github.com/junegunn/fzf.git ~/.fzf
~/.fzf/install

# Mac
brew install fzf
$(brew --prefix)/opt/fzf/install

# Windows (PowerShell)
winget install fzf
```

Built-in Key Bindings

```
Ctrl+R                # Fuzzy search command history
Ctrl+T                # Fuzzy find files in current directory
Alt+C                 # Fuzzy cd into subdirectories
```

Command-Line Usage

```
# Search through shell history
history | fzf

# Find and edit file
nano $(find . -type f | fzf)

# Select file to open
code $(ls | fzf)
```

```
# Interactive git branch switching
git switch $(git branch | fzf)

# Kill process interactively
kill $(ps aux | fzf | awk '{print $2}')

# Search and replace in files
rg TODO | fzf | cut -d: -f1 | xargs nano
```

Customization

```
# Add to ~/.bashrc or ~/.zshrc

# Customize appearance
export FZF_DEFAULT_OPTS='--height 40% --border --reverse'

# Use fd instead of find (faster, respects .gitignore)
export FZF_DEFAULT_COMMAND='fd --type f'

# Preview files with bat
export FZF_CTRL_T_OPTS="--preview 'bat --color=always {}'"

# Better directory preview
export FZF_ALT_C_OPTS="--preview 'tree -C {} | head -100'"
```

Advanced Examples

```
# Find and delete files
find . -type f | fzf -m | xargs rm

# Multi-select and copy files
cp $(find . -type f | fzf -m) /destination/

# Interactive process killer with preview
ps aux | fzf --header-lines=1 \
    --preview 'echo {}' \
    --preview-window down:3:wrap \
    | awk '{print $2}' | xargs kill
```

ripgrep (rg)

Installation

```
# Mac
brew install ripgrep

# Linux/WSL
sudo apt install ripgrep
```

```
# Windows
winget install ripgrep
```

Basic Usage

```
rg "search term"              # Search recursively from current dir
rg "pattern" path/            # Search in specific directory
rg -i "case-insensitive"      # Ignore case
rg -w "whole-word"            # Match whole words only
rg -n "show-line-numbers"     # Include line numbers (default)
```

File Type Filtering

```
rg "pattern" --type py        # Python files only
rg "pattern" -t js -t ts      # JavaScript and TypeScript
rg "pattern" -g '*.md'        # Markdown files only
rg "TODO" -g '!test*'         # Exclude test files
```

Advanced Searches

```
# Files containing pattern
rg -l "import pandas"         # Just list filenames

# Files NOT containing pattern
rg --files-without-match "TODO"

# Count matches per file
rg -c "error"

# Show context around matches
rg -C 3 "error"               # 3 lines before and after
rg -B 2 -A 4 "error"          # 2 before, 4 after

# Multiple patterns
rg "error|warning|critical"

# Search for exact string (not regex)
rg -F "function()"
```

Useful Combinations

```
# Search and edit
nano $(rg -l "TODO")

# Search with fzf preview
rg --line-number . | fzf --delimiter : \
    --preview 'bat --color=always {1} --highlight-line {2}'

# Find and replace across files
rg "old_text" -l | xargs sed -i 's/old_text/new_text/g'
```

Configuration

```
# Add to ~/.bashrc or ~/.zshrc

# Always show colors
alias rg='rg --color=always'

# Default to showing hidden files
export RIPGREP_CONFIG_PATH=~/.ripgreprc
# Then in ~/.ripgreprc:
--hidden
--glob=!.git/*
```

fd (Find Alternative)

Installation

```
# Mac
brew install fd

# Linux/WSL (might be 'fd-find')
sudo apt install fd-find
# Create alias: alias fd='fdfind'

# Windows
scoop install fd
```

Basic Usage

```
fd pattern                    # Find by name (case-insensitive)
fd '^test.*\.py$'             # Use regex
fd -e js                      # All JavaScript files
fd -e py -e ipynb             # Multiple extensions
```

Search Options

```
# Include hidden files/directories
fd -H config

# Include ignored files (.gitignore)
fd -I node_modules

# Search only files or only directories
fd -t f pattern               # Files only
fd -t d pattern               # Directories only
fd -t l                       # Symlinks only

# Search in specific directory
fd pattern /path/to/search
```

```
# Limit search depth
fd -d 3 pattern              # Max 3 levels deep
```

Execute Commands on Results

```
# Delete all .pyc files
fd -e pyc -X rm

# Move all .txt files
fd -e txt -x mv {} /destination/

# Run command for each result
fd -e js -x echo "Found: {}"

# Batch operations (passes all results at once)
fd -e log -X tar czf logs.tar.gz
```

Size and Time Filters

```
# Files larger than 100MB
fd . -t f -S +100m

# Files modified in last 24 hours
fd . --changed-within 24h

# Files modified between dates
fd . --changed-before 2024-01-01 --changed-after 2023-01-01

# Empty files
fd -t f -S 0
```

Useful Patterns

```
# Find all config files
fd -H '^\..*rc$'

# Find test files
fd test.*\.py$

# Find large directories
fd -t d -x du -sh {} | sort -rh | head -20

# Clean up cache files
fd -e cache -X rm
fd __pycache__ -X rm -rf
```

zoxide (Smart Directory Jumping)

Installation

```
# Mac
brew install zoxide

# Linux/WSL
curl -sS https://raw.githubusercontent.com/ajeetdsouza/zoxide/main/install.sh |
bash

# Windows
scoop install zoxide
```

Shell Setup

```
# Add to ~/.bashrc
eval "$(zoxide init bash)"

# Add to ~/.zshrc
eval "$(zoxide init zsh)"

# PowerShell: Add to $PROFILE
Invoke-Expression (& { (zoxide init powershell | Out-String) })
```

Basic Usage

```
z project                   # Jump to most frecent dir matching "project"
z proj                      # Partial matches work
z ~/Downloads proj          # Search within specific path
zi                         # Interactive selection with fzf
```

How It Works

- Frecency: Combines frequency (how often) + recency (how recent)
- Learns from your `cd` usage automatically
- The more you visit a directory, the higher it ranks
- Recent visits are weighted more heavily

Advanced Commands

```
# Query without jumping
zoxide query project        # See what would be matched
zoxide query --list         # List all tracked directories
zoxide query --score        # Show scores for ranking

# Add directory manually
zoxide add /path/to/dir

# Remove directory
zoxide remove /path/to/dir

# Edit database directly
zoxide edit
```

Useful Aliases

```
# Replace cd with z everywhere
alias cd='z'

# Quick directory stack
alias zz='z -'               # Toggle between last two dirs
```

Other Powerful Tools

Quick Reference

bat - Better Cat

```
brew install bat             # Install
bat file.txt                 # View with syntax highlighting
bat -n file.py               # Show line numbers
bat --theme=OneHalfDark     # Change theme
```

tldr - Simplified Man Pages

```
brew install tldr            # Install
tldr tar                     # Get practical examples
tldr -u                      # Update cache
```

httpie - User-Friendly HTTP

```
brew install httpie          # Install
http GET httpbin.org/get     # GET request
http POST api.com/data name=value  # POST with JSON
```

xan - CSV Power Tool

```
# Installation varies by system
xan select 1,3 file.csv      # Select columns
xan filter 'col1 > 100'      # Filter rows
xan sort -s col1 file.csv    # Sort by column
```

Terminal UI Tools

lazygit - Git TUI

```
brew install lazygit         # Install
lazygit                      # Launch in git repo
```

Key Shortcuts

- `Space` - Stage/unstage
- `c` - Commit
- `P` - Push
- `p` - Pull

- `x` - Show menu

lazydocker - Docker TUI

```
brew install lazydocker      # Install
lazydocker                   # Launch
```

Features

- View containers, images, volumes
- View logs in real-time
- Start/stop/remove containers
- Execute commands in containers

btop - System Monitor

```
brew install btop            # Install
btop                         # Launch
```

Features

- Beautiful CPU, memory, disk, network graphs
- Process management
- Mouse support
- Customizable themes

Configuration Management

Basic Git Setup for Dotfiles

Initial Setup

```
# Create dotfiles repository
mkdir ~/dotfiles && cd ~/dotfiles
git init

# Move existing configs
mv ~/.bashrc ~/dotfiles/bashrc
mv ~/.vimrc ~/dotfiles/vimrc
mv ~/.gitconfig ~/dotfiles/gitconfig

# Create symlinks
ln -s ~/dotfiles/bashrc ~/.bashrc
ln -s ~/dotfiles/vimrc ~/.vimrc
ln -s ~/dotfiles/gitconfig ~/.gitconfig

# Commit and push
git add .
git commit -m "Initial dotfiles commit"
git remote add origin <your-repo-url>
git push -u origin main
```

Restore on New Machine

```
# Clone repository
git clone <your-repo-url> ~/dotfiles
cd ~/dotfiles

# Create symlinks
ln -s ~/dotfiles/bashrc ~/.bashrc
ln -s ~/dotfiles/vimrc ~/.vimrc
ln -s ~/dotfiles/gitconfig ~/.gitconfig

# Reload shell
source ~/.bashrc
```

GNU Stow - Symlink Manager

Installation

```
# Mac
brew install stow

# Linux/WSL
sudo apt install stow
```

Directory Structure

```
~/dotfiles/
├── bash/
│   ├── .bashrc
│   └── .bash_profile
├── vim/
│   └── .vimrc
├── git/
│   └── .gitconfig
└── tmux/
    └── .tmux.conf
```

Using Stow

```
# Move configs into stow structure
mkdir -p ~/dotfiles/bash
mv ~/.bashrc ~/dotfiles/bash/

# Create symlinks
cd ~/dotfiles
stow bash                    # Links bash/.bashrc to ~/.bashrc
stow vim                     # Links vim/.vimrc to ~/.vimrc
stow git tmux               # Link multiple at once

# Remove symlinks
```

```
stow -D bash                 # Unstow (remove links)

# Check what would be linked
stow -n bash                # Dry run, no changes made
```

Advanced Stow Usage

```
# Custom target directory
stow -t ~ bash              # Explicitly set target

# Handle conflicts
stow --adopt bash           # Adopt existing files into dotfiles

# Re-stow (useful after changes)
stow -R bash                # Remove then re-add links
```

Quick Restore Script

Create install.sh

```
#!/bin/bash
# ~/dotfiles/install.sh

# Install dependencies
if [[ "$OSTYPE" == "darwin"* ]]; then
    # Mac
    brew install fzf ripgrep fd zoxide bat
elif [[ "$OSTYPE" == "linux-gnu"* ]]; then
    # Linux
    sudo apt update
    sudo apt install fzf ripgrep fd-find zoxide bat
fi

# Install fzf key bindings
~/.fzf/install --all

# Stow configurations
cd ~/dotfiles
stow bash vim git tmux

# Reload shell
source ~/.bashrc

echo "✓ Setup complete!"
```

Using the Script

```
chmod +x ~/dotfiles/install.sh
~/dotfiles/install.sh
```

Configuration Best Practices

Modular Config Files

```
# In ~/.bashrc
# Source additional configs
[ -f ~/.bash_aliases ] && source ~/.bash_aliases
[ -f ~/.bash_functions ] && source ~/.bash_functions
[ -f ~/.bash_local ] && source ~/.bash_local

# Keep machine-specific config in .bash_local (not in git)
```

Template for .bash_aliases

```
# Navigation
alias ..='cd ..'
alias ...='cd ../..'
alias ll='ls -lah'

# Git shortcuts
alias g='git'
alias gs='git status'
alias gd='git diff'
alias gc='git commit'
alias gp='git push'

# Safety
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# Tools
alias v='nvim'
alias lg='lazygit'
alias cat='bat'
```

Template for .bash_functions

```
# Create and enter directory
mkcd() { mkdir -p "$1" && cd "$1"; }

# Git commit with message
gcm() { git add . && git commit -m "$1"; }

# Find and edit
fe() { nano $(fzf); }
```

Quick Wins

Productivity Shortcuts

```
# Quick directory navigation
alias -- -='cd -'            # Go to previous directory
alias ~='cd ~'               # Go home
alias 1='cd -1'              # From directory stack
alias 2='cd -2'

# Git workflow
alias gaa='git add --all'
alias gcm='git commit -m'
alias gps='git push'
alias gpl='git pull'

# Quick edits
alias bashrc='nano ~/.bashrc && source ~/.bashrc'
alias aliases='nano ~/.bash_aliases && source ~/.bash_aliases'

# System monitoring
alias ports='netstat -tulanp'
alias psg='ps aux | grep -v grep | grep -i -e VSZ -e'

# Cleanup
alias cleanup='find . -type f -name "*.pyc" -delete'
alias rmds='find . -name ".DS_Store" -delete'
```

One-Liners to Try

```
# Find your most used commands
history | awk '{print $2}' | sort | uniq -c | sort -rn | head -10

# Find largest files in current directory
du -ah . | sort -rh | head -20

# Show directory sizes
du -sh */ | sort -rh

# Find files modified today
find . -type f -mtime -1

# Quick HTTP server for current directory
python3 -m http.server 8000

# Copy with progress bar
rsync -ah --progress source dest

# Monitor file changes
watch -n 1 'ls -lah'
```

Part 2 - Structured Data Processing

Get answers from APIs and datasets without leaving your terminal

jq - JSON Processing

Everything discussed here is a subset of common patterns with jq. Check out https://jqlang.org/manual/ for a full reference.

Install

```
sudo apt install jq           # Debian/Ubuntu/WSL
brew install jq               # macOS
winget install jqlang.jq      # Windows
```

Getting Fields

```
# Pretty-print any JSON
curl -s https://api.github.com/users/octocat | jq

# Get one field
curl -s https://api.github.com/users/octocat | jq '.location'
# "San Francisco"

# Raw string (no quotes) --- use -r whenever you're piping output onward
curl -s https://api.github.com/users/octocat | jq -r '.location'
# San Francisco

# Nested field
curl -s https://api.github.com/users/octocat | jq '.plan.name'

# Multiple fields --- build a new object
curl -s https://api.github.com/users/octocat | jq '{login, name, location}'
# {"login":"octocat","name":"The Octocat","location":"San Francisco"}
```

Working with Arrays

Most APIs return arrays. The `.[]` operator means “each item in the array.”

```
# Get a field from every item
curl -s https://api.github.com/users/octocat/repos | jq '.[].name'
# "boysenberry-repo-1"
# "git-consortium"
# "hello-worId"
#  ...

# Get a specific item by index
curl -s https://api.github.com/users/octocat/repos | jq '.[0].name'

# First/last N
curl -s https://api.github.com/users/octocat/repos | jq '.[:3]'   # First 3
curl -s https://api.github.com/users/octocat/repos | jq '.[-2:]'  # Last 2
```

```
# Count items
curl -s https://api.github.com/users/octocat/repos | jq 'length'
```

Filtering: `select()`

`select()` keeps items that match a condition.

```
# Repos that aren't forks
curl -s https://api.github.com/users/octocat/repos | \
  jq '.[] | select(.fork == false) | .name'

# Repos with stars
curl -s https://api.github.com/users/octocat/repos | \
  jq '.[] | select(.stargazers_count > 0) | .name'

# Combine conditions
curl -s https://api.github.com/users/octocat/repos | \
  jq '.[] | select(.fork == false and .stargazers_count > 0) | {name,
stars: .stargazers_count}'

# Posts by a specific user
curl -s https://jsonplaceholder.typicode.com/posts | \
  jq '.[] | select(.userId == 1) | .title'
```

Transforming Data

```
# Pick specific fields from each item
curl -s https://api.github.com/users/octocat/repos | \
  jq '.[] | {name, lang: .language, stars: .stargazers_count}'

# Collect results back into an array (wrap in [...])
curl -s https://api.github.com/users/octocat/repos | \
  jq '[.[] | {name, stars: .stargazers_count}]'

# Sort by field
curl -s https://api.github.com/users/octocat/repos | \
  jq 'sort_by(.stargazers_count) | reverse | .[:5] | .[].name'

# Get unique values
curl -s https://api.github.com/users/octocat/repos | \
  jq '[.[].language] | unique'

# Map: compute from every item
curl -s https://jsonplaceholder.typicode.com/todos | \
  jq 'map(select(.completed)) | length'

# Group by field
curl -s https://jsonplaceholder.typicode.com/posts | \
  jq 'group_by(.userId) | .[] | {userId: .[0].userId, posts: length}'
```

Output Formats

```
# Raw strings (no quotes) --- essential for piping to other commands
jq -r '.name' data.json

# Tab-separated
curl -s https://jsonplaceholder.typicode.com/users | \
  jq -r '.[] | [.name, .email] | @tsv'

# CSV
curl -s https://jsonplaceholder.typicode.com/users | \
  jq -r '.[] | [.name, .email, .phone] | @csv'

# Compact (one object per line)
cat data.json | jq -c '.[]'
```

Real Questions To Answer

“Which repos use which language?”

```
curl -s https://api.github.com/users/octocat/repos | \
  jq 'group_by(.language) | .[] | {lang: .[0].language, count: length}'
```

“Get the weather in a city”

```
curl -s "https://wttr.in/London?format=j1" | \
  jq '.current_condition[0] | {temp_C, humidity, desc: .weatherDesc[0].value}'
```

“List all TODOs for user 1 that aren’t done”

```
curl -s https://jsonplaceholder.typicode.com/todos | \
  jq '.[] | select(.userId == 1 and .completed == false) | .title'
```

“Extract all email addresses from an API”

```
curl -s https://jsonplaceholder.typicode.com/users | jq -r '.[].email' >
emails.txt
```

“Check user activity across multiple users”

```
for user_id in 1 2 3; do
  name=$(curl -s "https://jsonplaceholder.typicode.com/users/$user_id" | jq -r
'.name')
  post_count=$(curl -s "https://jsonplaceholder.typicode.com/posts?userId=$user_
id" | jq 'length')
  echo "$name: $post_count posts"
done
```

“Fetch paginated data”

```
# Loop through multiple pages
for page in {1..3}; do
```

```
  curl -s "https://jsonplaceholder.typicode.com/posts?_page=$page&_limit=10" | \
    jq -r '.[].title'
done
```

“Process each repo URL”

```
# Extract URLs and visit each one
curl -s https://api.github.com/users/octocat/repos | \
  jq -r '.[].html_url' | \
  while read -r url; do
    echo "Found repo: $url"
  done
```

“Conditional processing based on API response”

```
user=$(curl -s https://api.github.com/users/octocat)
repo_count=$(echo "$user" | jq '.public_repos')

if [ "$repo_count" -gt 20 ]; then
  echo "Prolific contributor: $repo_count repos"
else
  echo "Getting started: $repo_count repos"
fi
```

“Merge data from two API calls”

```
# Slurp (-s) reads multiple inputs into one array
curl -s https://jsonplaceholder.typicode.com/posts?userId=1 > /tmp/u1.json
curl -s https://jsonplaceholder.typicode.com/posts?userId=2 > /tmp/u2.json
jq -s 'add | length' /tmp/u1.json /tmp/u2.json
```

Using rg to Feed jq

When data isn’t pure JSON — buried in logs or mixed-format files — use `rg` to extract, then `jq` to process.

```
# Extract JSON objects from mixed log lines
rg -o '\{.*\}' app.log | jq -c 'select(.level == "ERROR")'

# rg can output structured JSON with --json --- process with jq
rg "TODO" --json | jq -r 'select(.type == "match") | .data.lines.text'

# Find all status codes in logged API responses
rg -o '"status":\d+' api.log | sort | uniq -c | sort -rn
```

DuckDB — SQL on Your Files

Install

```
sudo apt install duckdb       # Debian/Ubuntu/WSL
brew install duckdb           # macOS
winget install DuckDB.cli     # Windows
```

Query CSV Files Directly

```
# Start DuckDB shell
duckdb

# Or run one-off query from terminal
duckdb -c "SELECT * FROM 'sales.csv' LIMIT 10"
```

```
-- No CREATE TABLE, no import. Just query.
SELECT * FROM 'users.csv' LIMIT 5;

-- Filter
SELECT name, email FROM 'users.csv' WHERE age > 25;

-- Aggregate
SELECT department, COUNT(*) as headcount, AVG(salary) as avg_salary
FROM 'employees.csv'
GROUP BY department
ORDER BY avg_salary DESC;

-- Count rows
SELECT COUNT(*) FROM 'transactions.csv';
```

Query JSON Too

```
-- JSON array
SELECT * FROM 'data.json';

-- Newline-delimited JSON (one object per line)
SELECT * FROM read_json_auto('logs.jsonl');

-- Filter and aggregate
SELECT level, COUNT(*) as count
FROM read_json_auto('logs.jsonl')
GROUP BY level
ORDER BY count DESC;
```

Work with Remote Data

```
-- Query a CSV from a URL
SELECT * FROM 'https://raw.githubusercontent.com/datasets/covid-19/main/data/
countries-aggregated.csv'
WHERE Country = 'Germany'
ORDER BY Date DESC
LIMIT 5;

-- Combine multiple local files
SELECT * FROM 'logs/*.csv';
```

```
-- Glob patterns
SELECT * FROM 'data/2024-*.csv' WHERE amount > 1000;
```

Transform and Export

```
-- CSV to JSON
COPY (SELECT name, email FROM 'users.csv' WHERE active)
TO 'active_users.json' (FORMAT JSON);

-- JSON to CSV
COPY (SELECT * FROM 'data.json')
TO 'output.csv' (HEADER, DELIMITER ',');

-- One-liner from terminal
duckdb -c "COPY (SELECT * FROM 'big.csv' WHERE revenue > 10000) TO
'filtered.csv' (HEADER)"
```

Real Patterns

“Find duplicate emails”

```
SELECT email, COUNT(*) as n FROM 'users.csv'
GROUP BY email HAVING n > 1;
```

“Top 10 customers by total spend”

```
SELECT customer_id, SUM(amount) as total
FROM 'transactions.csv'
GROUP BY customer_id
ORDER BY total DESC
LIMIT 10;
```

“Join two CSV files”

```
SELECT o.order_id, c.name, o.total
FROM 'orders.csv' o
JOIN 'customers.csv' c ON o.customer_id = c.id
WHERE o.total > 100;
```

“Instant summary statistics”

```
SUMMARIZE 'sales.csv';
-- Shows count, min, max, avg, std, null% for every column
```

Nushell — A Different Way of Thinking

Everything above works in any shell. Nushell takes a different approach: **what if the shell itself understood structured data?**

In bash, every pipe passes text. In PowerShell, pipes pass objects. Nushell does the same — but for JSON, CSV, YAML, HTTP, and everything else.

```
# Bash + jq: repos with stars
curl -s https://api.github.com/users/octocat/repos | \
  jq '.[] | select(.stargazers_count > 100) | {name, stars: .stargazers_count}'

# Nushell
http get https://api.github.com/users/octocat/repos |
  where stargazers_count > 100 |
  select name stargazers_count

# Bash + jq: unique languages across repos
curl -s https://api.github.com/users/octocat/repos | \
  jq '[.[].language] | unique'

# Nushell
http get https://api.github.com/users/octocat/repos |
  get language |
  uniq

# Bash + jq: count completed TODOs per user
curl -s https://jsonplaceholder.typicode.com/todos | \
  jq 'map(select(.completed)) | group_by(.userId) | .[] | {userId: .[0].userId,
done: length}'

# Nushell
http get https://jsonplaceholder.typicode.com/todos |
  where completed == true |
  group-by userId |
  transpose userId todos |
  insert done {|r| $r.todos | length} |
  select userId done

# Bash + jq: extract emails to a file
curl -s https://jsonplaceholder.typicode.com/users | \
  jq -r '.[].email' > emails.txt

# Nushell
http get https://jsonplaceholder.typicode.com/users |
  get email |
  to text |
  save emails.txt

# Bash + jq: get weather as a quick summary
curl -s "https://wttr.in/London?format=j1" | \
  jq '.current_condition[0] | {temp_C, humidity, desc: .weatherDesc[0].value}'

# Nushell
http get "https://wttr.in/London?format=j1" |
  get current_condition.0 |
  select temp_C humidity weatherDesc.0.value
```

```
# Bash: count file types
find . -type f | rev | cut -d. -f1 | rev | sort | uniq -c | sort -rn

# Nushell
ls **/* | where type == file | get name | path parse | get extension | uniq --
count | sort-by count --reverse
```

```
# Process a CSV
open sales.csv | where region == "EU" | get amount | math sum

# Fetch, filter, convert, save
http get https://jsonplaceholder.typicode.com/users |
  where {|u| $u.address.geo.lat | into float | $in > 0} |
  select name email |
  to csv |
  save northern_users.csv
```

Using loops and conditionals

```
# Process each repo with a closure
http get https://api.github.com/users/octocat/repos |
  each {|repo|
    if $repo.stargazers_count > 50 {
      print $"($repo.name): ($repo.stargazers_count) stars"
    }
  }

# Fetch data from multiple pages
1..3 | each {|page|
  http get $"https://jsonplaceholder.typicode.com/posts?_page=($page)&_limit=10"
} | flatten

# Loop over files and process
ls *.json | each {|file|
  let data = (open $file.name)
  if ($data | length) > 100 {
    print $"Large file: ($file.name) has ($data | length) items"
  }
}

# Match expression for different cases
let response = (http get https://api.github.com/users/octocat)
match $response.public_repos {
  $n if $n > 50 => { print "Prolific contributor" }
  $n if $n > 10 => { print "Active developer" }
  _ => { print "Getting started" }
}
```

The tradeoff: nushell is a **shell**, not a tool. You don't pipe into it — you live in it. If that appeals to you, it replaces bash/zsh/PowerShell entirely. If you want something you reach for occasionally, jq and DuckDB are better fits.

Install and try it

```
brew install nushell          # macOS
winget install nushell        # Windows
# Or https://www.nushell.sh

nu                            # Start a nushell session
```

Quick Reference

jq Cheat Sheet

jq is vast and this is extremely incomplete. Remember to refer to https://jqlang.org/manual/ for a full reference.

```
jq '.'                    # Pretty-print
jq '.field'               # Get field
jq '.field.nested'        # Nested field
jq '.[0]'                 # Array index
jq '.[]'                  # Iterate array
jq 'length'               # Count items
jq 'keys'                 # Object keys
jq 'select(.x > 5)'       # Filter
jq 'map(.x)'              # Transform all items
jq 'sort_by(.age)'        # Sort
jq 'group_by(.type)'      # Group
jq 'unique'               # Deduplicate
jq '{name, age}'          # Pick fields
jq '[.[] | ...]'          # Collect into array
jq -r                     # Raw output (no quotes)
jq -c                     # Compact (one line)
jq -s                     # Slurp multiple inputs
jq '@csv'                 # Format as CSV
jq '@tsv'                 # Format as TSV
```

DuckDB One-Liners

```
duckdb -c "SELECT * FROM 'file.csv' LIMIT 10"
duckdb -c "SELECT col, COUNT(*) FROM 'f.csv' GROUP BY col ORDER BY 2 DESC"
duckdb -c "SUMMARIZE 'data.csv'"
duckdb -c "SELECT * FROM 'data.json'"
duckdb -c "SELECT * FROM '*.csv' WHERE status = 'active'"
duckdb -c "COPY (SELECT ...) TO 'out.csv' (HEADER)"
```

The Pipeline Pattern

Whatever tool you use, the thinking is the same:

GET	→ FILTER	→ TRANSFORM	→ AGGREGATE
curl	select()	{new fields}	length
cat	where	map()	group_by
open	.[]	@csv	sort_by
http get	SELECT WHERE	SELECT cols	COUNT, SUM

Get data in. Keep what matters. Reshape it. Summarize it.

Keep this guide handy - nobody memorizes all this!