

way of the hax0r
n00b to console punk

Chapter 4

Stop Fighting the Keyboard

By now you have typed a wrong character in the middle of a long command and retyped the whole line. You have also run a command, wanted it again with one change, and typed it from scratch. Both habits cost time, and both are unnecessary. The shell has a line editor, a memory of what you ran, and a way to finish names for you.

The concepts in this chapter are:

- the **line editor**, the part of the shell that handles what you type before Enter;
- **cursor movement** and **editing** keys, and which program owns each shortcut;
- **command history**, recall, and **history search**;
- **completion** of commands, paths, and options;
- **multiline input** and continuation prompts;
- **cancellation**: stopping a command, abandoning a line, and unfreezing a terminal.

Each shell has its own line editor with its own defaults. Bash uses **Readline**, zsh has the **Zsh Line Editor** (ZLE), and PowerShell uses the **PSReadLine** module. Command Prompt relies on the editing keys of the Windows console. Their default bindings overlap, which is convenient, but the overlap is a coincidence of history rather than a standard. [1], [2], [3]

Who Handles the Key You Pressed?

A keystroke travels through several programs, and the first one that recognizes it acts on it. Knowing which program owns a shortcut tells you where to look when it does not work and where to change it.

Owner	Examples	Where to change it
Terminal application	Ctrl+Shift+T new tab, Cmd+C copy on macOS, Ctrl+Shift+F search	Terminal settings
Terminal driver or console	Ctrl+C interrupt, Ctrl+Z suspend, Ctrl+D end of input, Ctrl+S pause output	stty on UNIX; console settings on Windows

Owner	Examples	Where to change it
Shell line editor	Ctrl+A, Ctrl+R, Tab, Up Arrow	Readline, ZLE, or PSReadLine configuration
Running program	q in a pager, Ctrl+X in nano, :q in Vim	The program's own settings

On UNIX-style systems the terminal driver, part of the operating system, turns Ctrl+C into an interrupt signal before the shell or program ever sees a character. `stty -a` prints those assignments; look for `intr`, `susp`, `eof`, `stop`, and `start`. [4] On Windows, the console host performs the same role for Ctrl+C. [5]



Trap: Ctrl+S on a UNIX terminal pauses output, and the terminal looks frozen. Nothing is broken; the terminal driver is waiting for Ctrl+Q to resume. Press Ctrl+Q before assuming a program has hung.

Everything below describes default bindings. Your configuration, or a tutorial's, may differ; Chapter 10 shows how to change them.

Move the Cursor

The line editor keeps your command in a buffer until you press Enter. Enter runs the whole buffer no matter where the cursor is, which you saw in Chapter 1.

Bash/zsh

Readline and ZLE start in **Emacs mode**, named after the editor whose keys they borrow. [2], [6]

Action	Keys
Start or end of line	Ctrl+A, Ctrl+E; also Home, End in most terminals
One character	Left, Right; also Ctrl+B, Ctrl+F
One word	Alt+B, Alt+F



Wut: zsh switches to its **vi** keymap by itself when the `EDITOR` or `VISUAL` environment variable contains `vi`. If Ctrl+A inserts a character instead of moving the cursor, that is why. `bindkey -e` restores the Emacs keys for the session. [2]

Alt here means the Readline **Meta** key. Pressing Esc and then the letter does the same thing when Alt is taken by the terminal. macOS Terminal sends Alt+B as a special character unless you enable **Use Option as Meta key** in the profile's keyboard settings. [7]

PowerShell

PSReadLine has two default **edit modes**, and the mode decides most of the keys in this chapter. On Windows the default is **Windows mode**; on macOS and Linux it is **Emacs mode**. [3]

Action	Windows mode	Emacs mode
Start or end of line	Home, End	Ctrl+A, Ctrl+E, also Home, End
One word	Ctrl+Left, Ctrl+Right	Alt+B, Alt+F

Get-PSReadLineOption reports the current mode in its EditMode property, and Get-PSReadLineKeyHandler lists every binding. Check the mode before following a tutorial's key list. [8]

Command Prompt

The console provides Home, End, Left, Right, and Ctrl+Left and Ctrl+Right for word movement. There is no Ctrl+A binding for start of line. [9]

Edit the Line

Deleting a few characters with Backspace is fine. Deleting a whole argument or the rest of a line is faster with a **kill** command, which removes text and keeps it so you can **yank** it back.

Bash/zsh

Action	Keys
Delete to start of line	Ctrl+U
Delete to end of line	Ctrl+K
Delete the word before the cursor	Ctrl+W
Delete the word after the cursor	Alt+D
Paste the last deleted text	Ctrl+Y
Clear the screen, keeping the line	Ctrl+L
Undo the last edit	Ctrl+_ or Ctrl+X Ctrl+U

Try it with a line you do not want to run. Type `echo one two three`, press Ctrl+W twice, then Ctrl+Y twice, and read the result before you press Enter. [10]

PowerShell

Action	Windows mode	Emacs mode
Delete to start of line	Ctrl+Home	Ctrl+U
Delete to end of line	Ctrl+End	Ctrl+K
Delete the word before the cursor	Ctrl+Backspace	Ctrl+W
Paste the last deleted text	Ctrl+Y	Ctrl+Y
Undo	Ctrl+Z	Ctrl+_
Abandon the line	Esc	Ctrl+C

In Windows mode Ctrl+A selects the whole line, as in a text editor, and Ctrl+C copies a selection when there is one. [3], [8]

Command Prompt

Esc clears the line. Ctrl+Home and Ctrl+End delete to the start and end of the line. There is no yank. [9]

History

The shell records the commands you run. Up and Down Arrow walk through them, and each recalled command can be edited before you press Enter.

More useful than walking is **searching**. Press Ctrl+R, type part of a command, and the shell shows the most recent match. Press Ctrl+R again for an older match, Enter to run it, or the left or right arrow to edit it first. This binding is the same in Readline, ZLE, and both PSReadLine modes. [2], [3], [11]

Bash/zsh

history prints the numbered list. Bash also expands **history designators** before running a command:

- !! repeats the previous command;
- !\$ is the last argument of the previous command;
- !42 runs command number 42.

Run echo hello and then echo !! to see the expansion appear on your screen before the second command runs. zsh has the same designators. [11], [12], [13]



Trap: History expansion happens inside double quotes. echo "Surprise!" is fine because ! is followed by a quote, but echo "Hello!world" tries to expand !world. Use single quotes for text containing !, or type set +H in Bash to turn expansion off for the session.

Where history is saved, and how much, depends on the shell's configuration. Bash writes to ~/.bash_history when an interactive shell exits by default; zsh saves nothing until HISTFILE is set, which most distributions and macOS do in a startup file. Chapter 10 returns to these settings. [14], [15]

PowerShell

Get-History lists this session's commands with numbers, and Invoke-History 5 or its alias r 5 reruns one. PSReadLine keeps its own history file across sessions, separately from Get-History, and it is the one that Up Arrow and Ctrl+R use. [3], [16]

In Windows mode, F8 searches backward for commands that start with what you have typed so far.

Command Prompt

Up and Down Arrow recall commands within the session. F7 shows a menu of the session's history, and doskey /history prints it. The list is not saved when the window closes. [9]

Completion

Typing the first few characters of a name and pressing Tab asks the line editor to finish it. Completion reads the real filesystem and command list, so it is also a way to check spelling without running anything.

Bash/zsh

Press Tab once to complete a unique name. If several names match, Bash completes the common prefix and a second Tab lists the choices. zsh lists the choices immediately and, with its completion system enabled, lets you cycle through them with repeated Tab. [17], [18]

Try `cd ~/Do` followed by Tab and see what your home directory offers. On Linux, most distributions install **bash-completion**, which adds completion for command options and arguments. Without it, Bash completes command names and paths only.

PowerShell

In Windows mode, Tab cycles through the matches one at a time, and Shift+Tab cycles backward. In Emacs mode, Tab behaves like Bash and lists the choices. In Windows mode, Ctrl+Space opens a **menu completion**, where arrow keys pick a match; Chapter 10 shows how to bind MenuComplete in other modes. [3]

PowerShell completes parameter names too. Type `Get-ChildItem -` and press Tab to walk through the parameters of that cmdlet. It also completes `$env:` variable names and drive names.

Command Prompt

Tab cycles through matching file and directory names in the current directory. It does not complete command names or options. [19]

More Than One Line

A long command can be split across lines, and some constructs require it. The shell shows a **continuation prompt** while it waits for the rest.

Bash/zsh

A backslash at the very end of a line continues the command on the next line. Bash shows > as the continuation prompt by default:

```
$ echo one \  
> two  
one two
```

An unclosed quote also continues the line, which is why a missing quotation mark leaves the shell waiting. Press Ctrl+C to abandon the whole thing. [20]

PowerShell

PowerShell continues automatically when a line cannot be complete, such as after an open brace, a pipe character, or a comma. A backtick at the end of a line continues it explicitly. The continuation prompt is `>>`: [21]

```
$ Write-Output one `
>> two
one
two
```

Note that `Write-Output` received two arguments and printed each on its own line, which is a difference from the Bash example, not a mistake. `Shift+Enter` inserts a new line inside the buffer without running it, in both edit modes.

Command Prompt

A caret at the end of a line continues the command, and the continuation prompt is `More?`:

```
$ echo one ^
More? two
one two
```

The caret is Command Prompt's escape character, which you saw in Chapter 2. [19]

Cancel, Stop, and Quit

Four different situations look similar and need different keys.

- **You typed a line you do not want.** `Ctrl+C` in Bash, `zsh`, and PowerShell abandons it and gives you a fresh prompt. `Esc` does the same in Command Prompt and in PowerShell's Windows mode.
- **A command is running and you want it to stop.** `Ctrl+C` asks the program to stop. Most programs comply; Chapter 9 covers the ones that do not.
- **A program is waiting for input and you have finished.** `Ctrl+D` on UNIX and `Ctrl+Z` then `Enter` in Command Prompt send **end of input**. At an empty Bash or `zsh` prompt, `Ctrl+D` exits the shell, which surprises people.
- **A program has taken over the screen.** Pagers exit with `q`, nano with `Ctrl+X`, and Vim with `Esc` then `:q`. Chapters 5 and 6 cover them.

`Ctrl+Z` at a UNIX prompt does not undo anything. It suspends the running program, which Chapter 9 explains; if you press it by accident, `fg` brings the program back.

Key Points

- Each shell has its own line editor with its own defaults: Readline, ZLE, PowerShell, or the Windows console.
- A shortcut belongs to the terminal, the terminal driver, the shell, or the running program, and that owner is where to look when it does not work.

- `Ctrl+A`, `Ctrl+E`, `Ctrl+U`, `Ctrl+K`, `Ctrl+W`, and `Ctrl+Y` edit the line in Emacs-mode editors; PSReadLine's Windows mode uses editor-style keys instead.
- `Ctrl+R` searches history in every shell covered here except Command Prompt.
- `Tab` completes names, and the shells differ in whether they list or cycle.
- Backslash, backtick, and caret continue a line in Bash/zsh, PowerShell, and Command Prompt.
- `Ctrl+C` cancels, `Ctrl+D` ends input, `Ctrl+Z` suspends on UNIX, and `Ctrl+Q` unfreezes a terminal paused by `Ctrl+S`.

Level Up

n00bs finish the chapter here. You can edit a line without retyping it, find an old command, and complete names instead of spelling them. Hackers start bending the editor to their habits.

Print your editor's full binding list and read it: `bind -p` in Bash, `bindkey` in zsh, and `Get-PSReadLineKeyHandler` in PowerShell. Find three bindings you did not know and try them.

Each editor has a **vi mode** that uses Vim's keys instead of Emacs keys: `set -o vi` in Bash, `bindkey -v` in zsh, and `Set-PSReadLineOption -EditMode Vi` in PowerShell. Try it for a session after Chapter 5 introduces Vim, and decide which mode fits you.

Investigate **fzf**, a fuzzy finder that replaces `Ctrl+R` with an interactive search over history and can complete paths the same way. Explain what it adds beyond the built-in search, and what it costs in setup.

Look at PSReadLine's **predictive IntelliSense**, which suggests commands from history as you type, and at zsh's `zsh-autosuggestions` plugin, which does the same. What data do they use, and where is it stored?

Exercises

1. **What does this do?** In Bash, type `echo alpha beta gamma`, then press `Ctrl+W`, `Ctrl+A`, `Ctrl+Y`. Predict the line before you look.
2. **What does this do?** Run `echo hello` and then `echo !!` in Bash. Explain each line of output, including the one the shell prints before running the second command.
3. **What is the problem?** A PowerShell user on Windows presses `Ctrl+A` expecting the cursor to jump to the start of the line, and the whole line is highlighted instead. Explain what happened and give two ways to get the cursor movement they wanted.
4. **What is the problem?** A terminal stops responding after a user tries to save with `Ctrl+S`. Nothing appears, but the program is not hung. Explain what happened and how to fix it.
5. **Why is it like this?** `Ctrl+C` is copy in most desktop programs but cancel in a terminal. Explain which program decides, and why the terminal applications in Chapter 1 use `Ctrl+Shift+C` for copy.
6. **Why is it like this?** `Get-History` in PowerShell shows only this session's commands, but `Up Arrow` recalls commands from last week. Explain which component owns each list.
7. **How do you** rerun the command before last with one argument changed, in Bash, without retyping it? Give two methods.
8. **How do you** find out which key deletes the word before the cursor in your current PowerShell edit mode, without reading this chapter?

References

- [1] GNU Project, “Bash reference manual: Command line editing.” Available: https://www.gnu.org/software/bash/manual/html_node/Command-Line-Editing.html
- [2] Zsh Development Group, “The z shell manual: Zsh line editor.” Available: <https://zsh.sourceforge.io/Doc/Release/Zsh-Line-Editor.html>
- [3] Microsoft, “About_PSReadLine.” Available: https://learn.microsoft.com/en-us/powershell/module/psreadline/about/about_psreadline
- [4] GNU Project, “GNU coreutils manual: Stty: Print or change terminal characteristics.” Available: https://www.gnu.org/software/coreutils/manual/html_node/stty-invocation.html
- [5] Microsoft, “Console control handlers (windows console).” Available: <https://learn.microsoft.com/en-us/windows/console/console-control-handlers>
- [6] GNU Project, “Bash reference manual: Readline interaction.” Available: https://www.gnu.org/software/bash/manual/html_node/Readline-Interaction.html
- [7] Apple, “Terminal user guide: Change profiles keyboard settings.” Available: <https://support.apple.com/guide/terminal/change-profiles-keyboard-settings-trmlkbrd/mac>
- [8] Microsoft, “Get-PSReadLineKeyHandler.” Available: <https://learn.microsoft.com/en-us/powershell/module/psreadline/get-psreadlinekeyhandler>
- [9] Microsoft, “Windows commands: doskey.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/doskey>
- [10] GNU Project, “Bash reference manual: Bindable readline commands.” Available: https://www.gnu.org/software/bash/manual/html_node/Bindable-Readline-Commands.html
- [11] GNU Project, “Bash reference manual: History expansion.” Available: https://www.gnu.org/software/bash/manual/html_node/History-Interaction.html
- [12] GNU Project, “Bash reference manual: Bash history builtins.” Available: https://www.gnu.org/software/bash/manual/html_node/Bash-History-Builtins.html
- [13] Zsh Development Group, “The z shell manual: expansion.” Available: <https://zsh.sourceforge.io/Doc/Release/Expansion.html>
- [14] GNU Project, “Bash reference manual: Bash history facilities.” Available: https://www.gnu.org/software/bash/manual/html_node/Bash-History-Facilities.html
- [15] Zsh Developers, “The z shell manual: parameters.” Available: <https://zsh.sourceforge.io/Doc/Release/Parameters.html>
- [16] Microsoft, “Get-history.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/get-history>

- [17] GNU Project, “Bash reference manual: Letting readline type for you.” Available: https://www.gnu.org/software/bash/manual/html_node/Commands-For-Completion.html
- [18] Zsh Development Group, “The z shell manual: Completion system.” Available: <https://zsh.sourceforge.io/Doc/Release/Completion-System.html>
- [19] Microsoft, “Windows commands: cmd.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/cmd>
- [20] GNU Project, “Bash reference manual: quoting.” Available: https://www.gnu.org/software/bash/manual/html_node/Quoting.html
- [21] Microsoft, “About_special_characters.” Available: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_special_characters