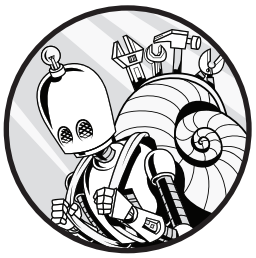
1

FLYING ON THE COMMAND LINE



Being able to rapidly move around the shell and manipulate text on the command line is critical to being an effective shell user. As you spend more time in the shell and start composing larger and more complex commands, it's especially important that you can work efficiently. In this chapter, we'll look at some techniques to help you do just that. Before long, you'll be navigating the command line at lightning speed.

To see this chapter's examples in action, go to *https://effective-shell.com*, where you'll find animated images of each technique. I encourage you to also try them out as you go along.

Basic Navigation Techniques

In this section, we'll look at a number of shortcuts you can use to maneuver the cursor. To begin, use the following command to write the quote to a text file:

```
$ echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >> note.txt
```

Once you have executed this command by pressing ENTER, the quote will be written to a file called *note.txt*.

The shortcuts introduced in this chapter allow you to move around and manipulate the command line much more quickly and efficiently than if you were using only the arrow keys or delete key.

To work through the examples in this section, press the up arrow key, which will bring the command back up in your shell with the cursor at the end of the line.

Go to the Beginning or End of a Line

You can quickly jump to the beginning of the text, no matter where your cursor is currently positioned, with CTRL-A:

```
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >> note.txt
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >> note.txt
```

The shortcut CTRL-E takes you to the end of the line:

```
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >> note.txt
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >> note.txt
```

These are two of the most useful shortcuts, and I highly recommend that you incorporate them into your regular shell usage.

Move Back or Forward One Word

You can also quickly jump backward or forward one word at a time. Use ALT-B to move back one word:

```
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >> note.txt
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >> note.txt
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >> note.txt
```

In this example, using ALT-B once takes you back one word to the start of `txt`. Using it a second time takes you to the start of `note`. The shell uses the dot (`.`) character as a word separator; therefore, `note` and `txt` are treated as two separate words (they could also be separated by a space or other non-alphanumeric symbol).

To go back to the beginning of the line (if you are not there already), use CTRL-A:

```
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >> note.txt
```

Now use ALT-F to go forward one word at a time:

```
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >> note.txt
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >> note.txt
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >> note.txt
```

Notice that these shortcuts not only jump over each space between the words but also differentiate between symbols and actual words. In this example, the cursor moves from the beginning of the line to the end of the word `echo`, which is the space that follows it. The space and the quotes that follow `echo` are two sequential characters that are not part of a word, so the next time ALT-F is pressed, the cursor moves to the end of the next word (the space after `When`).

Delete a Word

To quickly delete a word, place the cursor at the end of it and use CTRL-W. If your cursor is at the end of the line as in the following example, pressing CTRL-W would work like so:

```
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >> note.txt
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >>
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin
echo "When you light a candle, you also cast a shadow." - Ursula Le
```

If the cursor is halfway through a word, CTRL-W will delete only from the beginning of the word to the cursor position:

```
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >> note.txt
echo "When you light a candle, you also cast a dow." - Ursula Le Guin >> note.txt
```

To delete the next word or character to the right, use ALT-D:

```
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >> note.txt
echo "When you light a , you also cast a shadow." - Ursula Le Guin >> note.txt
echo "When you light a also cast a shadow." - Ursula Le Guin >> note.txt
```

Notice that CTRL-W defines a word as any nonspace character, whereas ALT-D defines a word as only alphanumeric characters. This is because under the hood different "delete word" functions are available in the shell, and the default keyboard shortcuts use different variants for deleting a word and deleting the next word.

You can look up how each shortcut works by running `man bash` to open the man page for bash and searching for "commands for moving."

Delete a Line

In bash, you can delete everything from the current cursor position to the beginning of the line with CTRL-U:

```
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >> note.txt
Ursula Le Guin >> note.txt
```

NOTE *If you're using the Z shell, this key combination will delete the entire line regardless of where your cursor is. If you're not sure what shell you're using, look at the prompt next to your cursor. If it starts with a hash mark (#) or dollar sign ($), it's probably bash. If it starts with a percent symbol (%), it's probably the Z shell. On most Linux machines, your shell will be bash by default. On macOS systems from 2019 onward, it will be the Z shell. To see exactly what shell you have, run* ***echo $SHELL****.*

To delete everything from the cursor to the end of the line, use CTRL-K:

```
echo "When you light a candle, you also cast a shadow." - Ursula Le Guin >> note.txt
echo "When you light a candle, you also cast a shadow."
```

This should work the same way in bash and the Z shell.

Undo a Change

Bash also has an undo shortcut: CTRL-_ (underscore) will undo the most recent change.

If you find yourself repeatedly using the arrow or delete keys, refer back to this section to remind yourself of the shortcuts. They'll save you a lot of time in the long run!

Search Commands

Once you have the basic navigation commands down, the next essential shortcuts are search commands. Starting with the current line, you can search backward or forward in your command history with CTRL-R and CTRL-S, respectively.

Let's look at an example. Run these commands on three separate lines to create nine empty files:

```
$ touch file1 file2 file3
$ touch file4 file5 file6
$ touch file7 file8 file9
```

Now press CTRL-R to start searching, and you'll see a search prompt in your shell. Because you're searching backward, the prompt tells you that you're doing a reverse search. At the prompt, enter **file** and press CTRL-R repeatedly:

```
(reverse-i-search)`file': touch file7 file8 file9
(reverse-i-search)`file': touch file7 file8 file9
(reverse-i-search)`file': touch file7 file8 file9
(reverse-i-search)`file': touch file4 file5 file6
```

The shell will search the previous commands for the term `file`, jumping farther back each time you press CTRL-R. This is quite hard to visualize

in printed text, so be sure to try it out in your shell. When the shell reaches the end of the search and can't find any more `file` entries, the prompt will change to something like:

```
(failed reverse-i-search)
```

Press ENTER to get back to the regular prompt. Searching forward with CTRL-S works in much the same way.

If your code lines are long, these two shortcuts are often the fastest ways to move to the desired location in the current line. You can also use them to quickly search through your entire command history. For example, to find your last `mkdir` ("make directory") command, press CTRL-R, enter **`mkdir`**, and then press CTRL-R again to search backward through all `mkdir` commands stored in your history.

A quick way to test this is also to search for `echo`. If you've been using the `echo` command to enter the quote as in the earlier examples, it should be the first result that the reverse search finds.

When you find the command you want, just press ENTER to execute it. If you want to edit the command first, use the left or right arrow keys to go back into normal editing mode. This would be useful if, say, you want to rerun a long *`commandname`* command but on a different file. You can search for *`commandname`* and then change the filename.

If you want to cancel the search completely, press CTRL-G. The search prompt will disappear, and whatever you had in the command prompt before you started searching will be returned.

Editing in Place

When working with a long or complex command, you might find it easier to use a text editor instead of the shell. You could just copy the command, paste it into your favorite editor, edit it, and then paste it back into the shell. However, in most cases you can open your text editor right inside the shell.

The default editor for the shell is often set to Vim, which you'll see in detail in Chapter 23. Before going any further, set your editor to nano, which is more user-friendly, like so:

```
$ export EDITOR=nano
```

If this command doesn't make sense at the moment, don't worry—it will soon. You'll also learn more about customizing shell features such as the default editor in Chapter 15.

Now that you've set the default editor to nano, you need to enter two shortcut combinations to open it: first, press CTRL-X to signal to the shell that you're about to enter a command, and then press CTRL-E to edit in place. If you don't have any unexecuted code on your command line, the text editor will be empty when it opens; otherwise, it will show your current command.

Let's see this in action. Begin entering a command to write the list of programs in your */usr/bin* folder to a file:

```
$ ls -al /usr/bin >> binaries.txt
```

But instead of pressing ENTER to execute this code, press CTRL-X, CTRL-E to edit it in place (see Figure 1-1).

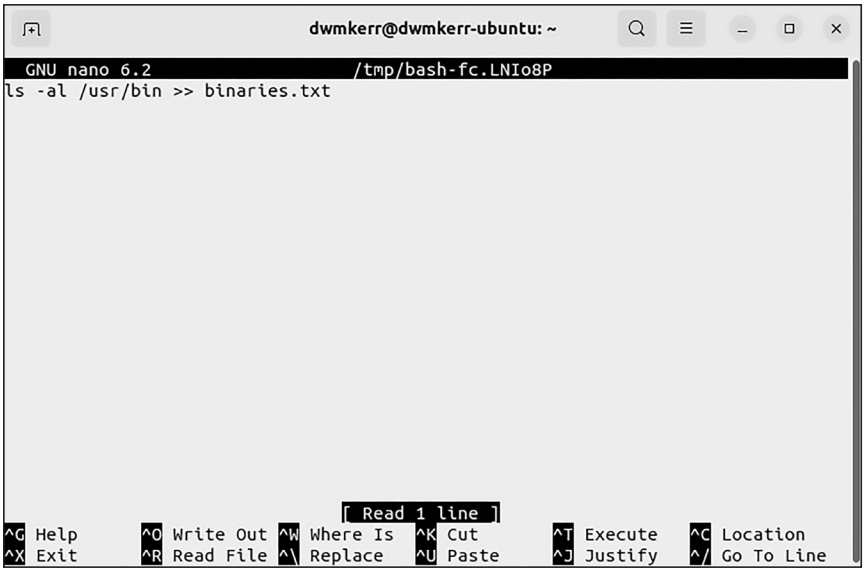


Figure 1-1: The "edit in place" functionality in nano

The nano editor uses keyboard shortcuts to save, close, cut, copy, and paste, and so on. These shortcuts are shown at the bottom of the screen. You won't be able to click any buttons or menus.

Now you can edit the command text by deleting characters and typing new ones. Save your changes with CTRL-S, and exit the editor with CTRL-X. The shell will then run the edited command. You can see the most recent command by pressing the up arrow. If you want to discard your changes without running the command, close the editor without saving.

NOTE *If the editor does not look like the screenshot in Figure 1-1, you may have opened Vim by mistake. If so, type* **`:q!`** *and press ENTER to exit Vim, then enter* **`export EDITOR=nano`** *to change the default editor for your shell.*

Keep in mind that the CTRL-X, CTRL-E shortcut opens the shell's *default* editor, which might not be the one you expect. For example, your shell is unlikely to use Visual Studio Code for editing commands unless you configure it to do so, even if you have set it as your editor for shell scripts. The default editor will be one that works *inside* a shell, because the shell doesn't assume that you have a windowing system running that can open a full-featured editor like Visual Studio Code, Notepad++, or other popular editors.

As you saw at the beginning of this section, you can override the shell's default editor by setting the `EDITOR` environment variable (see Chapter 10 for more on variables). On my personal machine, I use Vim as my command line editor. To check which editor you're using, enter the following:

```
$ echo $EDITOR
vim
```

For now, I wouldn't recommend changing your editor to a graphical one like Visual Studio Code for a couple of reasons. First, some interfaces, like a virtual machine or Raspberry Pi, won't have a desktop system to run a graphical editor, so you'll need to be familiar with a shell-based editor. Second, an external graphical editor runs in a separate window, meaning you've moved out of the shell and away from where you're working. Effective shell users want as few interruptions as possible between coming up with an idea, entering the text, and running a command.

Sometimes graphical editors can come in handy, though! We'll look at some more advanced editors in Chapter 23.

Other Useful Shortcuts

There are a few other shortcuts that, while they don't fit into the categories we've looked at so far, are just as handy.

Clear the Screen

The shortcut CTRL-L clears the screen but doesn't affect anything unexecuted in your current line. This is very helpful if you have a lot of "noisy" output on the screen and want to clean it up.

View Your Command History

Running the `history` command prints the recent history of commands you've entered:

```
$ history
    1  pwd
    2  ls
    3  git status
    4  clear
    5  curl effective.sh | sh
    ...
```

You'll get a numbered list of your command history, with the latest at the bottom. By default, the shell saves about 10,000 lines of history.

NOTE *This command works even if you've used CTRL-L to clear the screen.*

If you want to rerun any of the commands in your history, enter an exclamation mark (!) and the command's number like so:

```
$ !5
effective-shell: preparing to install the 'effective-shell.com' samples...
```

The 5 corresponds to the command `curl effective.sh | sh`, which installs the *effective-shell* samples. Most shells maintain your command history in a history file, and this number is just the line number from that file. You can find the history file's location by running:

```
$ echo $HISTFILE
/home/dwmkerr/.bash_history
```

Where the history file is kept depends on your shell, configuration, and operating system, but in most cases the `HISTFILE` variable will find it for you.

Show All Shortcuts

The `bindkey` command returns a list of all keyboard shortcuts:

```
$ bindkey
"^@" set-mark-command
"^A" beginning-of-line
"^B" backward-char
"^D" delete-char-or-list
"^E" end-of-line
"^F" forward-char
"^G" send-break
"^H" backward-delete-char
"^I" expand-or-complete
"^J" accept-line
"^K" kill-line
"^L" clear-screen
...
```

This is an extremely useful command if you forget a specific keyboard shortcut or even if you just want to see the shortcuts available to you. If `bindkey` doesn't work for you, try using the alternative form `bind -p`.

Transpose Text

Transposing text just means swapping it with some other text. Using the ALT-T shortcut transposes the two words before the cursor:

```
$ cp destination source
$ cp source destination
```

Using the CTRL-T shortcut will transpose the two *letters* before the cursor.

Summary

In this chapter, you learned shortcuts for maneuvering your cursor in the shell, which allows you to edit your code quickly and easily. You also saw how to list and search your command history, edit in place with the shell's default text editor, and take advantage of other techniques to work more efficiently on the command line.

In the next chapter, we'll look at how to find files and folders on your system, a process that is often complex and time-consuming even in a GUI environment but is a snap with the shell.