The Readline Library

In this lesson, you'll learn about what readline is, what bash looks like without it, and how to manipulate and find out more about its features.

WE'LL COVER THE FOLLOWING

Readline is one of those technologies that is so commonly used people don't realise it's there. In this lesson I want to make you aware of it and introduce some concepts and examples of its use so you're able to deal with any problems that arise from its use or misuse.

How Important is this Lesson?

Writing bash does not require a knowledge of readline. Understanding readline helps greatly to understand what is going on at the terminal and at the command line.

Bash Without Readline

First you're going to see what bash looks like without readline.

In your 'normal' bash shell, hit the `TAB` key twice. You should see something like this:

```
Display all 2335 possibilities? (y or n)
```

That's because bash normally has an *autocomplete* function that allows you to see what commands are available to you if you tap `TAB` twice.

Hit `n` to get out of that autocomplete.

Similarly, if you hit the up arrow key a few times, then the previously-run commands should be brought back to the command line.

Now type:

```
bash --noediting
```

Type the above code into the terminal in this lesson.

● Terminal

The `--noediting` flag starts up bash without the readline library enabled. If you hit tab twice now you will see something different: the shell no longer 'sees' your tab and just sends a tab direct to the screen. Autocomplete has gone.

Autocomplete is just one of the things that the readline library gives you in the terminal. You might want to try hitting the up or down arrows as you did above to see that that no longer works as well.

Hit return to get a fresh command line, and exit your non-readline-enabled bash shell:

```
exit
```

Type the above code into the terminal in this lesson.

Other Shortcuts

There are a great many shortcuts like autocomplete available to you if readline is enabled. I'll quickly outline four of the most commonly-used of these before explaining how you can find out more.

```
echo 'some command'
```

Type the above code into the terminal in this lesson.

There should not be many surprises there. Now:

- If you hit the `up` arrow, you will see you can get the last command back on your line
- If you like, you can re-run the command, but there are other things you can do with readline before you hit return
- If you hold down the `ctrl` key and then hit `a` at the same time your cursor will return to the start of the line
- Another way of representing this 'multi-key' way of inputting is to write it like this: `\C-a`. The `\C` represents the control key, and the '`-a`' represents that the `a` key is depressed at the same time
- Now if you hit `\C-e` (`ctrl` and `e`) then your cursor has moved to the end of the line. I use these two shortcuts dozens of times a day
- Another frequently useful one is `\C-l`, which clears the screen, but leaves your command line intact
- The last one I'll show you allows you to search your history to find matching commands while you type. Hit `\C-r`, and then type `ec`. You should see the `echo` command you just ran like this:

```
(reverse-i-search)`ec': echo echo
```

- Then do it again, but keep hitting `\C-r` over and over
- You should see all the commands that have `ec` in them that you've input before (if you've only got one `echo` command in your history then you will only see one)
- As you see them you are placed at that point in your history and you can move up and down from there or just hit return to re-run if you want

Here's a table of the most frequently used shortcuts for reference:

Readling Shortcut	Effect

\C-a	Takes you to start of line
\C-e	Takes you to end of line
\C-h	Delete character
\C-l	Clear screen
\C-r	Review history

> Note: What is your **history**? Your history is a list of commands that were previously-ran in your shell. We will cover this in a later lesson.

There are many more shortcuts that you can use that readline gives you. Next I'll show you how to view these.

Using `bind` to Show Readline Shortcuts

If you type:

```
bind -p
```

Type the above code into the terminal in this lesson.

You will see a list of bindings that readline is capable of. There's a lot of them! Have a read through if you're interested, but don't worry about understanding them all yet.

If you type:

```
bind -p | grep C-a
```

Type the above code into the terminal in this lesson.

you'll pick out the 'beginning-of-line' binding you used before, and see the `\C-a` notation I showed you before.

As an exercise at this point, you might want to look for the `\C-e` and `\C-r` bindings we used previously.

If you want to look through the entirety of the `bind -p` output, then you will want to know that:

- `\M` refers to the `Meta` key (which you might also know as the `Alt` key)
- `\e` refers to the `Esc` key on your keyboard. - The 'escape' key bindings are different in that you don't hit it and another key at the same time, rather you hit it, and then hit another key afterwards
    - For example, typing the `Esc` key, and then the `?` key also tries to auto-complete the command you are typing. This is documented in the `bind -p` output as:

```
"\e?": possible-completions
```

Readline and Terminal Options

If you've looked over the possibilities that readline offers you, you might have seen the `\C-r` binding we looked at earlier:

```
"\C-r": reverse-search-history
```

You might also have seen that there is another binding that allows you to search forward through your history too:

```
"\C-s": forward-search-history
```

What often happens to me is that I hit `\C-r` over and over again, and then go too fast through the history and fly past the command I was looking for. In these cases I might try to hit `\C-s` to search forward and get to the one I missed.

Watch out though! Hitting `\C-s` to search forward through the history might well not work for you.

Why is this, if the binding is there and readline is switched on?

It's because something picked up the `\C-s` before it got to the readline library:

It's because something picked up the `\C-s` before it got to the readline library: the terminal settings.

The terminal program you are running in may have standard settings that do other things on hitting some of these shortcuts before readline gets to see it.

If you type:

```
stty -a
```

Type the above code into the terminal in this lesson.

you should get output similar to this:

```
speed 38400 baud; rows 24; columns 101; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = M-^?; eol
2 = M-^?; swtch = <undef>;
start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V; dis
card = ^O;
min = 1; time = 0;
-parenb -parodd -cmspar cs8 hupcl -cstopb cread -clocal -crtscts
-ignbrk brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ix
off -iuclc ixany imaxbel
iutf8
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt
0 ff0
isig icanon iexten echo echoe echok -echonl -noflsh -xcase -tostop -echopr
t echoctl echoke -flusho
-extproc
```

You can see on the second line there is a list of key bindings that your terminal will pick up before readline sees them. The `^` character (known as the *caret*) here represents the '`ctrl`' key that we previously represented with a `\C`.

If you think this is confusing I won't disagree. Unfortunately, in the history of Unix and Linux documenters did not stick to one way of describing these keys.

If you encounter a problem where the terminal options seem to catch a shortcut key binding before it gets to readline, then you can use the `stty` program to unset that binding. In this case, we want to unset the 'stop' binding.

If you are in the same situation, type:

```
stty stop undef
```

Type the above code into the terminal in this lesson.

Now, if you re-run `stty -a`, the list of key bindings should look like this:

```
[...]
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = M-^?; eol
2 = M-^?; swtch = <undef>;
start = ^Q; stop = <undef>; susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^
V; discard = ^O;
```

where the `stop` entry now has ’`<undef>` assigned to it.

Strangely, for me `C-r` is also bound to ‘reprint’ above (`^R`). But (on my terminals at least) that gets to readline without issue as I search up the history. Why this is the case I haven’t been able to figure out. I suspect that reprint is ignored by modern terminals that don’t need to ‘reprint’ the current line.

While we are looking at this list:

```
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = M-^?; eol
2 = M-^?; swtch = <undef>;
start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V; dis
card = ^O;
```

it’s worth noting a few other key bindings that are used regularly.

First, one you may well already be familiar with is `\C-c`, which interrupts a program, terminating it:

```
sleep 99 # NOW Hit \C-c
```

Type the above code into the terminal in this lesson.

After hitting `\C-c` you should see the output:

```
^C
```

Similarly, `\C-z` suspends a program, allowing you to ‘foreground’ it again and continue with the fg builtin.

```
sleep 10 # NOW hit \C-z
fg
sleep 10
```

Type the above code into the terminal in this lesson.

`\C-d` sends an ‘end of file’ character. It’s often used to indicate to a program that input is over. If you type it on a bash shell, the bash shell you are in will close.

Finally, `\C-w` deletes the word before the cursor

These are the most commonly-used shortcuts that are picked up by the terminal before they get to the readline library.

Readline Quiz

1 `\e` refers to the

COMPLETED 0%

1 of 3

What You Learned

- What the *readline* library is

- What the *readline* library is
- What features it gives you
- How to find out more about those features
- The order in which terminal options and *readline* are considered
- Some terminal-native shortcuts
- How key combinations are represented in standard documentation

What Next?

Armed with your increased knowledge of how the *readline* library works, and terminal program settings, you are ready to tackle the tricky subject of **terminal codes**.

Exercises

1) Research the terminal options' shortcuts in the `stty -a` list and what they mean.

2) Research the inputrc file and edit it to configure how `backward-kill-word` is invoked.