The Prompt

In this lesson, you will learn how to set up your main prompt, as well as other types of prompts and related shell variables.

WE'LL COVER THE FOLLOWING

Now that you've learned about escapes and special characters you are in a position to understand how the bash **prompt** can be set up and controlled.

How Important is this Lesson?

This lesson is not essential, but most people find it interesting and maybe fun to learn about.

The `PS1` Variable

Type this:

```
bash
PS1='My super prompt>>>>   '
ls
exit
```

Type the above code into the terminal in this lesson.

Terminal

As you'll remember, there are some *shell variables* that are set within bash that are used for various purposes. One of these is `PS1`, which is the prompt you see after each command is completed.

The `PS2` Variable

```
bash
PS2='I am in the middle of something!>>>  '
cat > /dev/null << END
some text
END
exit
```

Type the above code into the terminal in this lesson.

The PS2 variable is the 'other' prompt that the shell uses to indicate that you are being prompted for input to a program that is running. By default, this is set to `> `, which is why you see that as the prompt when you normally type the `cat` command above in.

`PS3`

`PS3` is used by the `select` looping structure. We don't cover that in this book as I've barely ever seen it used.

`PS4`

`PS4` is the last one:

```
bash
PS4='> Value of PWD is: $PWD'
set -x
pwd
cd /tmp
ls $(pwd)
cd -
exit
```

Type the above code into the terminal in this lesson.

In 'trace' mode `PS4` is echoed before each line of trace output. Do you

remember what trace mode is?

But why is the `>` in the output repeated? This indicates the *level* of indirection (ie subshells) in the trace. Every time the script goes one level 'down' a shell, the first character in the `PS4` value is repeated. Look at the output after the `ls $(pwd)` command.

> Note: Things can get really confusing if you have commands in your prompt, or you have `PROMPT_COMMAND` set (see below section). If you don't fully understand the output of the above, don't panic!

Pimp Your Prompt

For all the `PS` variables mentioned above, there are special escape values that can be used to make your prompt display interesting information.

See if you can figure out what is going on here:

```
bash
PS1='\u@\H:\w \# \$ '
ls
exit
```

Type the above code into the terminal in this lesson.

The table below may help you understand:

```
Escape value | Meaning                | Notes
-------------------------------------------------------------------
\#           | Command number         | The number (starting
             |                        | from 1 and incrementing
             |                        | by one) of the command
             |                        | in this bash session.
-------------------------------------------------------------------
\$           | Root status            | If you have root, show a
             |                        | '#' sign, otherwise show
             |                        | '$'
-------------------------------------------------------------------
\t           | Current time           | In HH:MM:SS format -
             |                        | there are other formats
             |                        | possible with eg \T.
-------------------------------------------------------------------
```

```
\H            | Hostname                    | The hostname
              |                             | (fully-qualified)

-------------------------------------------------------------------
\w            | Current working             |
              | directory                   |
-------------------------------------------------------------------
\[            | Start control sequence      | Begin a sequence of
              |                             | non-printing characters,
              |                             | eg put a terminal
              |                             | control sequence in a
              |                             | prompt.
```

Use your knowledge gained so far to figure out what is going on here:

```
bash
PS1='\[\033[01;31m\]PRODUCTION\$ '
PS1='\[\033[01;32m\]DEV\$ '
exit
```

Type the above code into the terminal in this lesson.

Line 1 starts a fresh bash shell to work in so you don't 'taint' the existing shell with an overwritten prompt variable.

Line 2 is an example of setting the prompt to a control sequence that sets the colour of the prompt to red (`31m`) for a 'production' server.

Line 3 is an example of setting the prompt to a control sequence that sets the colour of the prompt to green (`32m`) for a 'production' server.

Line 4 exits the freshly-created bash shell and returns you to your original prompt.

You might want to try different numbers instead of `31` and `32` above to see what colours are set, and research which colours map to which numbers with an online search.

How would you make this automatically happen on a given server when you log in?

Show Hint

The `PROMPT_COMMAND` Variable

Another way the prompt can be affected is with the bash variable `PROMPT_COMMAND`:

```
bash
PROMPT_COMMAND='echo "Hello prompt $(date)"'
ls
exit
```

Type the above code into the terminal in this lesson.

Every time the prompt is displayed the `PROMPT_COMMAND` is treated as a command, and run.

You can use this for all sorts of neat tricks!

What You Learned

- What the `PS` variables are
- Where each `PS` variable is used
- How to augment your *prompts* to give you useful information
- How to automatically run commands before each *prompt* is shown

What Next?

Next you will learn a very useful technique for quickly creating files: the **here doc**.

Exercises

1) Look up the other prompt escape characters and use them.

2) Update your bash startup files so that the prompt tells you useful information when you log in

3) Create your own version of the `history` command by using the `PROMPT_COMMAND` variable.