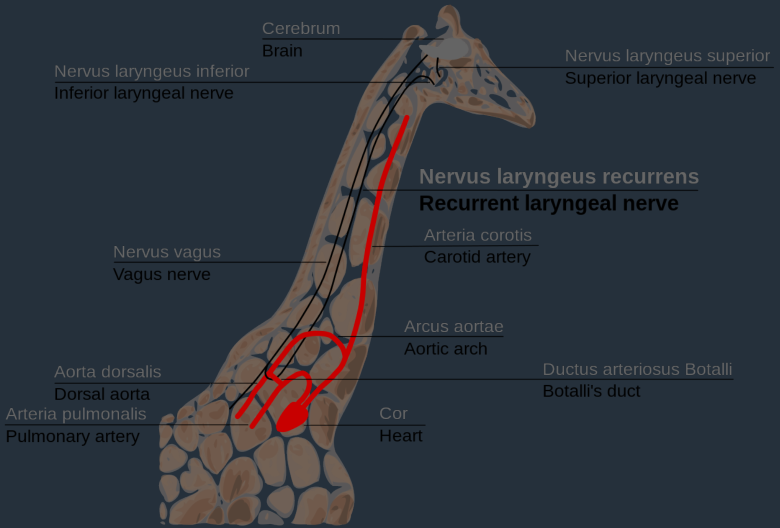
Using the terminal effectively

Peter Hill

Outline

- Escape codes
- Customising the prompt
- The command line and readline
- History
- Command, process and variable substitutions
- Aliases and functions

Terminals are old



https://commons.wikimedia.org/wiki/File:GiraffaRecurrEn.svg

Terminals are old

https://commons.wikimedia.org/wiki/File:Teletype.jpg

Fancier terminals

- konsole
- terminology
- terminator
- guake
- tilda
- rxvt-unicode
- xterm
- cool-retro-term

Escape codes

- Also known as control characters
- "In-band signalling"
- Terminal would intercept these and do something else instead of printing them
- Cover things like backspace, ringing the bell, newline, etc.
- Also allowed setting text attributes: bold, underscore, different colours
- Because they aren't designed for printing, they might be hard to type, or look a bit odd. Many include the `ESC` character (hence the name):

```
\033[030m          ^[[30m
ESC [ 3 0 m        \e[30m
```

- "`^[`" is the code for `C-[`, which is also `ESC` or `\e` (0x1b, 033 in octal)
- Actually many different types of terminals, that support different control character sets. We're normally interested in "xterm-256color" and "ANSI" escape sequences
  - Look under `/usr/share/terminfo` for a few other examples...

Using colours

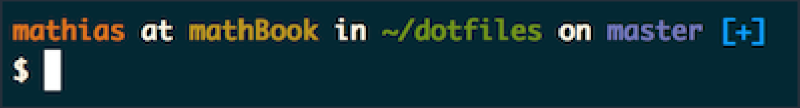
- Set foreground colour with "\033[03<0-8>m", and reset with "\033[039m"
- Set background colour with "\033[04<0-8>m", and reset with "\033[049m"
- Normally just put all the colours into variables and reference them:

```
WARN_COLOUR="\033[031m"
RESET_COLOUR="\033[039m"
echo -e "${WARN_COLOUR}WARNING: badness${RESET_COLOUR}"
```

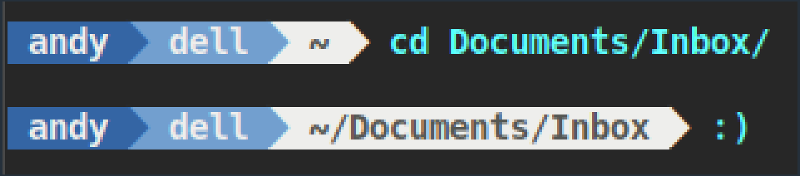
- Can use these colours in anything that writes to terminal (even Fortran!)

```
character(len=*), parameter :: red = char(27) // "[031m"
character(len=*), parameter :: reset = char(27) // "[039m"
print*, red // "WARNING: badness" // reset
```

Customising the prompt

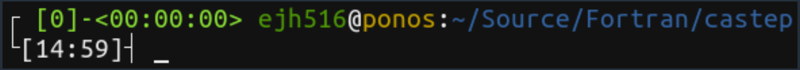


```
mathias at mathBook in ~/dotfiles on master [+]
$ █
```

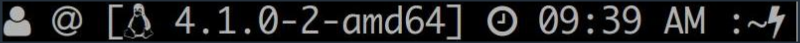


```
 andy  dell  ~  cd Documents/Inbox/

 andy  dell  ~/Documents/Inbox  :)
```



```
┌ [0]-<00:00:00> ejh516@ponos:~/Source/Fortran/castep
└[14:59]┤ _
```



```
 @ [ 4.1.0-2-amd64]  09:39 AM :~
```

Customising the prompt

PS1

- Default value is `\s-\v\$`
- Lots of options: `info bash -n Controlling` to see full list
- `[\t] \u@\h \w:` turns into `[15:27:30] user@hostname ~/directory:`
- To use colours, we need to surround them with an additional `\[` and `\]`
- This lets bash know that they won't take any space up on screen

PROMPT_COMMAND

- This is a command that is run every time before displaying the prompt
- You can use this to show you information about e.g the git repo you are in, or the number of jobs you have running on a supercomputer

Movement on the command line

- `readline` is the secret hero here
- Readline provides many, many commands for moving about on the command line
- `info readline` to find out more
- Follow the basic Emacs commands
- `C-` means “Ctrl”, `M-` means “Alt” (used to be “Meta”)
- `C-a`/`C-e`: move to beginning/end of line
- `M-f`/`M-b`: move forward/backward by a word
- `Shift-PgUp`/`Shift-PgDown`: scroll backwards/forwards

GNOME is annoying

- In GNOME, the default terminal grabs the Alt key
- Turn this off: Edit > Keyboard Shortcuts. . . , uncheck “Enable menu access keys”

Editing commands

- `M-d` to delete the following word
- `C-k` to delete from the cursor to the end of the line
- `C-u` to delete from the cursor to the beginning of the line
  - Also works in lots of other places in Linux!
- `M-#` to comment out a line
- Fix a mistake on the previous line by running `^a^b^` to replace the first instance of "a" with "b" and then rerun the command
  - Also useful for rerunning a command with a different parameter
- If a command is becoming long and hard to edit, you can open it in your `$EDITOR` with `C-x C-e`
  - For Emacs, the best thing to do is set `$EDITOR` to `emacsclient` and `M-x start-server` in Emacs – this will then cause things to pop-up in your existing Emacs session

Magic of readline

Quick aside

- You can use readline in your own programs
- You can even use readline to wrap other programs that don't support it out of the box – `rlwrap` https://github.com/hanslub42/rlwrap
- For python projects, also check out `prompt-toolkit` https://github.com/jonathanslenders/python-prompt-toolkit

Movement through history

- Search with `C-r`
- You can also enable a fancier search. Put the following in your `~/.inputrc`:

```
"\e[A": history-search-backward
"\e[B": history-search-forward
```

- Reload your inputrc with `C-x C-r`
- Now you can start typing a previous command and then use the cursor keys to browse all commands that start with those letters:

```
./ma...
./magic
./magical
```

Working out keycodes

Quick aside

- Quickest way to work out what keycode to put is to run `sed -n l` then hit the key and press enter:

```
sed -n l
^[[A
```

History expansion

- Special variables for referring to previous commands, all start with "`!`"
  - This is why you might struggle to use "`!`" in commands/strings
- `!!`: Repeat the previous command
- `!N`: Refer to command on line N
- `!-N`: Refer to the command N lines back
- `!foo`: Refer to the last command starting with "foo"
- `!$`: Use the value of the last argument from the previous command
- You can also insert the last argument from the previous command with `M-.`
  - Except on Macs, where you need to do `ESC-.`, or change how `option` works
  - You can also prefix with a number: `M-2 M-.` to get the second argument (with zero being the previous command)

Keeping history

The problem with multiple terminals

- If you use multiple terminals, their histories get out of sync
- By default, only the history from last one open is kept!
- Easy fix: append to the history file on every command:

```
shopt -s histappend
PROMPT_COMMAND="history -a"
HISTFILESIZE=1000000000
HISTSIZE=1000000
```

- Last two commands just make sure we keep a lot of history. . .

Tab completion

- Hit `TAB` to auto-complete commands and filenames
- maybe you're lazy like me, and don't care about capitalisations in filenames, etc. Put the following in your `~/.inputrc`:

```
set completion-ignore-case On
TAB: complete
"\e[Z": menu-complete
```

- Super useful when traversing the filesystem!

Command substitution

- Use the output of one command in another one: `$(command)`
  - You can also use backticks, but `$()` is better
- Nest them!

```
echo $(ls $(echo foo))
```

Actually useful example

```
which pip
less $(!!)
```

- Find out where a command is installed (is it a system package, or something I've installed myself?)
- Assuming I think it's a script, have a look at its contents

Process substitution

Another way of joining programs together

- How to compare the output of running two different programs?
- Could just dump the output of each program into separate files and then `diff` them
  - This is boring
- Better way is “process substitution”:

```
diff <(command1) <(command2)
diff <(command1 | sort | uniq) <(command2 | sort | uniq)
diff <(ssh archer 'cat remote/file') local_file
```

- Connects the output of the “inner” commands with the input argument of the “outer” command

Variable substitution

- Bash has some fancy uses for curly braces:
- Drop the extension from a filename: `${foo%.*}`
- Or replace it with a different one: `${foo/tex/pdf}`
- Get the length of a string: `${#foo}`
- Read more: http://wiki.bash-hackers.org/syntax/pe

Curly brace expansion

- Quick way to iterate over a few options: `{a,b,c}` gives `a b c`
- `a{b,c}d` gives `abd acd`
- Useful for installing multiple packages:
  - `sudo apt install {lapack,hdf5}-dev`
  - will install both the lapack and hdf5 development packages
- Copying one file to another:
  - `cp filename{,.bak}`
- Also does ranges: `{1..10}` gives numbers 1 to 10, `{a..z}` gives...

Aliases

- Aliases are “another name” for a command
- Useful if you always run a command with the same options

`ls` family

```
alias ls='ls -hF --color'    # add colors for filetype recognition
alias la='ls -Alh'           # show hidden files
alias lt='ls -ltrh'          # sort by date, most recent last
```

Functions

- Use functions for more complicated expressions
- If you find yourself writing particularly complicated bash, stop! Use a better language instead!

Useful example

```
function latest() {
    # Print the most recent file in a given directory
    lastfile=$(ls -tc --color=tty "$@" | head -1);
    echo "$@$lastfile";
}

# Move the last file I downloaded here
mv -v "$(latest ~/Downloads)" .
```

Find idioms

Different ways of grepping files from `find`

```
find path/ -type f -exec grep foo {} \;
find path/ -type f | xargs grep foo
for f in $(find path/ -type f); do grep foo $f; done
```

Different shells

- ksh if you want more POSIX
- zsh if you want to be like Ed
- fish if you want to really stand out
- tcsh if you want to die inside
- xonsh if you really, really like python

Further reading

- http://wiki.bash-hackers.org/scripting/terminalcodes
- https://en.wikipedia.org/wiki/GNU_Readline
- `info readline`
- https://stackoverflow.com/a/1862762/2043465
- http://wiki.bash-hackers.org/syntax/pe
- https://github.com/alebcay/awesome-shell