Using the shell

Johan Öhlin

4 November 2021

Yabs

Good to know

Commonly used keyboard shortcuts:

- Previously typed line: $\uparrow$ or C+p
- Other way around: $\downarrow$ or C+n
- Moving back and forward on line:
  - Backward/forward one letter: $\leftarrow$ / $\rightarrow$ or C+b / C+f
  - Backward/forward one word: A+b / A+f
  - Start of line: C+a
  - End of line: C+e
- Search for used command with C+r
- Clear screen with C+l
- Execute with C+o or C+j

Good to know

Commonly used keyboard shortcuts:

- Deleting characters
  - Remove previous letter: C+h
  - Remove next letter: C+d
  - Remove* previous word: C+w
  - Remove* next word: A+d
  - Remove* line: C+u
  - Remove* rest of line: C+k
- ***Actually** they cut. Use C+y to paste what you just removed, but only the last key-press.

Good to know

Commonly used keyboard shortcuts:

- Search for used command with C+r (back) or C+s (forward)
- Clear screen with C+l
- Execute with C+o or C+j
- Cancel a command with C+c
- Terminate command or shell with C+d

Aliasing

- You can make use of aliases if you are lazy and don't want to type as much.
- Make some guards for commands.
- Make (short) scripts.
- Make a `command` out of your script.

Aliasing

To show your current aliases, only type `alias`.

```
$ whatis alias
alias:  nothing appropriate.
```

Note: Different shells will have different ways of creating aliases.

- `alias l='ls -Cf'`
- `alias rm='rm -i'`
- `alias ab='ls -p | wc -b; whoami'`
- `alias xy='/path/to/script --with-flags'`
- `alias listfiles='ls'`
- `alias listfiles='l'`

See if you have any aliases already, and try to create new ones.

Aliasing & .bashrc

- To make aliases permanent, i. e. be available each time you start a new instance, you can keep it a bash startup file.
- By default, you Bash startup file will be located in *~/.bashrc*. **Note:** Files starting with . are *hidden*, in the regard that `ls` will not show them by default. You might have many various startup, configuration files and directories in your home directory. To see those files, you run `ls -a`.

Aliasing & .bashrc

- The startup file will be loaded each time you start a new shell. If you wish to load this startup in an already opened shell, you can type `source ~/.bashrc`.
- You will later learn about scripts and how to make functions. You startup may also contain similar functions which you can use as commands.

You can now try to edit your startup file.

Cygwin users: To locate your startup file, run

```
$ realpath ~/.bashrc | xargs cygpath -w
```

.bashrc & variables

Other than aliases and functions, your startup file can set variables which you might find useful.

These can be set in the shell straight, but will be no kept between sessions.

```
$ HW="Hello world"
$ echo $HW
```

Your shell has plenty of these so-called environment variables already. Those can be viewed with `env`

To make environment variables you will have the variables *exported*.

To export, use command `export`

```
$ export HW="Hello world"
```

.bashrc & variables

```
$ echo $HW

$ HW="Hello World"
$ echo $HW
Hello World
$ env | grep HW
$ export HW
$ env | grep HW
HW=Hello World
$ export HW="Bye World"
HW=Bye
```

.bashrc & variables

Variables within variables:

```
$ A="Hello"
$ b="World"
$ C="$A $b"
$ C="$C $C"
$ echo $C
Hello World Hello World
```

Can be good practice to use (e. g.) `$HOME` variable instead of hard-coded path to *your* home directory.

Note: If you have a program which looks at the environment variables, you can override it as `$ VAR=yes env`

.bashrc & functions

Result of command in variable.

```
$ A=$(ls)
$ echo $A
file1
file2
$ echo $A | head -n1
file1 file2
$
```

Note: Different shells will handle this differently.

.bashrc & functions

You can have functions in your startup file instead of aliases.

```
$ cat ~/.bashrc
function notls {
  echo "This is not $VAR"
  $VAR | cat -n
}
VAR="ls"
```

Note: This (above) can written directly into your shell.

Functions

Write in to your shell. The indentations are not important.

```
$ function notls {
  echo "This is not $VAR"
  $VAR | cat -n
}
```

Run `notls` in shell. Set `VAR` and run again:

```
$ VAR="ls"
```

Change `VAR` and run again:

```
$ VAR="whoami"
```

Functions

You can call functions from functions.

```
$ function highlightk{
  # Colour the letter k
  notls | grep --color 'k'
}
```

.bashrc & functions

To use arguments:

```
$ function highlightk{
  # Colour $1 from notls
  notls | grep --color "$1"
}
```

functions

Warining: You can recursively run you function. Just make sure to avoid *fork bombs*.

```
$ function f {
    f | f &
}
```

&

To start a process in the background, you can end the line with one single ampersand.

```
$ sleep 30 &
```

To see which processes you have started, use the command `jobs`.

```
$ jobs
[2]- Kör sleep 20 &
```

You can set it as a foreground process again with the command `fg`. `$ fg`

&

To make a foreground process into a background process, you have to stop the process. This is achieved with C-z.
You can later make it into a background process with the command `bg`.
The process will still be connected to you shell. To dispatch it, use `disown`.

```
$ jobs
$ sleep 20
^Z
[1]+ Stoppad sleep 20
$ bg
[1]+ sleep 20 &
$ # Wait for 20 seconds
[1]+ Klart sleep 20
$
```