Tips

Clear and reset the terminal

To clear the terminal use the command below or press Ctrl+L on your keyboard.

```
>_ clear
```

If you screw up the screen by e.g. accidentally reading a binary file with with cat or less you might reinitialisation the terminal with the reset command.

```
>_ reset
```

List of recent commands

Use the history command to list all recent commands.

```
>_ history
```

Search command history for query.

```
>_ history | grep [query]
```

or press ctrl + r to search and execute commands from the history.

Close a frozen window/application

Execute the command and click on the frozen window.

```
>_ xkill # x11 window
```

Or find the process id of an application and kill it.

```
>_ ps ax | grep firefox<br> [processID] ?? S 0:00.22 firefox<br> >_ kill [processID]
```

Tab Completion

This might save you a lot of time. If you e.g. want to delete a file with a very long name you can type the first few characters of the name and press TAB to auto complete the name. If there are more then one possibilities and you press TAB twice you get a list of all possibilities.

Redirection

You can redirect standard input, output and error by adding these sequences after a command.

> send output to file (overwriting and destroying whatever is in the file already)

```
>_ echo "Write output to file" &gt; example.txt
```

>> append output to file

```
>_ echo "Append this to file" &gt;&gt; example.txt
```

< take input from file

```
>_ cowsay &lt; example.txt
```

2> send error messages to file (overwriting). (This means that errors can be directed separately from normal output.)

```
>_ rm -vf folder1 file1 > out.txt 2> err.txt
```

View file permissions

```
>_ ls -l [file]
```

1 = File type: \'-\' for a regular file, \'d\' for a directory, \'l\' for a symbolic link.
2 = Owner permission
3 = Group permission
4 = Other permission
In the shown example the user has read, write and execute permission but the group and other has only read permissions.

Modify file permissions

```
>_ chmod [modification] [file]
```

Example: Give the group read and write access to the test.txt file.

```
>_ chmod g+rw test.txt
```

Permission groups:
u = Owner
g = Group
o = Other
Permission types:
r = Read
w = Write
x = Execute
Operators for the modification command are + (plus) and – (minus).

Set file permissions via binary references

Example: Give the owner read and write permission, the group read permission and no permission to other.

```
>_ chmod 640 test.txt
```

The first number represents the **owner** permission, the second the **group** permissions and the last number for all other users. The numbers are a binary representation of the **rwx** string.

r = 4
w = 2
x = 1

Cursor navigation

ctrl + u = Clear everything before the cursor
ctrl + a = To beginning of line
ctrl + e = To end of line
ctrl + b = Back one word
ctrl + f = Forward one word
ctrl + w = Cut last word
ctrl + k = Clear everything after cursor
ctrl + _ = Undo

Special characters in commands

Shell special characters are interpreted by the shell as soon as it is given the command. For example, if you type ls *.bak, the shell translates *.bak to the list of all files in the current folder whose names end in .bak. The ls command never sees the asterisk. So if you want to search for files which actually have an asterisk in their names, you have to escape the asterisk to stop the shell from interpreting it.

\ escapes itself and other specials

***** stands for anything (including nothing)

```
>_ find ex*.txt
```

? stands for any single character

```
>_ find ex?mple.txt
```

[] encloses patterns for matching a single character

```
>_ find ex[abc]mple.txt
```

() runs the contents of the parentheses in a sub-shell

```
>_ pwd &amp;&amp; ( cd /etc) &amp;&amp; pwd<br> /home/simon<br> /home/simon
```

; terminates a command pipeline - use it to separate commands on a single line

```
>_ echo Hi ; uname<br> Hi<br> Linux
```

' ' The contents of the single quotes are passed to the command without any interpretation.

>_ find '(echo abc)'*<br>(echo abc).txt

` ` The contents of the backquotes are run as a command and its output is used as part of this command

>_ echo ` uname`<br>Linux

" " The contents of the quotes are treated as one argument; any specials inside the quotes, except for $ and ``, are left uninterpreted.

>_ cd "untitled folder"

| Pipes allow you to send the output of a command to another command.

>_ fortune | cowsay

& Run a command in the background.

>_ cowsay &amp;

&& Only execute the second command if the first one was successful.

>_ ping localhost -c 1 &amp;&amp; cowsay great

|| Only execute the second command if the first one was unsuccessful.

>_ ping "not.reachable" -c 1 || cowsay sorry

>> These symbols are used for redirection.

!! Repeat the last command

>_ sudo !!

!* Change command keep all arguments

>_ head history | grep query<br> >_ !* tail<br> tail history | grep query

^ Quick history substitution, changing one string to another.

>_ ls *.png<br>toast.png<br>>_ ^png^xcf^<br> ls *.xcf<br>bread.xcf

Turns the line into a comment; the line is not processed in any way.

>_ whatis xdotool # hint: has sth todo with X11

Don't confuse shell special characters with special characters in regular expressions. Regular expressions must be protected from the shell by enclosing them in single quotes.