"Mastering the Terminal" Cheat Sheet

1. Opening and Closing the Terminal

Opening The Terminal	CTRL + ALT +T
Closing The Terminal	CTRL + D

2. Basic Commands

echo	Prints command line arguments to standard output.
date	Show the current date and time.
cal	Display a calendar.
cat	Stick files together and write joined file to standard output. Good for viewing the contents of 1 file.

3. Command History

history	Show commands previously entered (command history).
!!	Run the previous command.
!50	Run the command that is on line 50 of the output from the history command. (replace "50" as needed).

NB: history –c; history –w; will clear the history of commands.

4. Some Important Definitions

Command	An instruction typed in the terminal and submitted to the shell for interpretation.
Shell	A program that interprets commands for meaning.
Terminal	A graphical window where commands can be typed and submitted to the shell.

5. Command Structure

Each command follows the same overarching structure:

```
commandName –options arguments
```

5.1 Command Names

commandName must be a valid program on the Shell's Path. To check this, use the which command like so:

```
which commandName
```

If a path is returned, then the commandName is valid and vice versa.

5.2 Options

You can specify options for each command to customise the commands behaviour. These can be either "short-form" options or "long-form" options.

Each command behaves differently so check the command's manual (man) page for the specifics of each command's behaviour.

5.2.1 Short-form Options

Short-form options are where a letter defines an option. Each option is prepended by a dash "-" like so:

```
commandName –a –b –c args
```

To save typing, you could join together the options:

```
commandName –abc args
```

Both of these formats are equivalent.

5.2.2 Long-form Options

For some commands, there are long-form options defined to make options easier to identify. Longform options are usually prepended by a double dash "--".

Long-form options **cannot** be joined together like short-form options can.

Whether they are defined or not depends on each specific command, so consult the command's manual page for more information.

If long form options are defined for options "a", "b" and "c", then:

```
commandName –a –b -c arguments
```

is equivalent to

```
commandName --alpha --beta --charlie arguments
```

5.2.3 Command Line Arguments

Command line arguments are a type of input that commands operate on.

Some commands can take an unlimited amount of inputs, some take a specific amount, and some take none at all. Consult the manual page for the specific command for more information.

```
cal 12 2017
```

Here the cal command has 2 command line arguments. The number 12 and the number 2017.

5.2.4 Arguments for Options

Sometimes, command options can also take their own arguments (inputs).

```
cal –A 1 –B 1 12 2017
```

Here the cal command has 2 options; A and B.

The A option has its own argument (1).

The B option has its own argument (1).

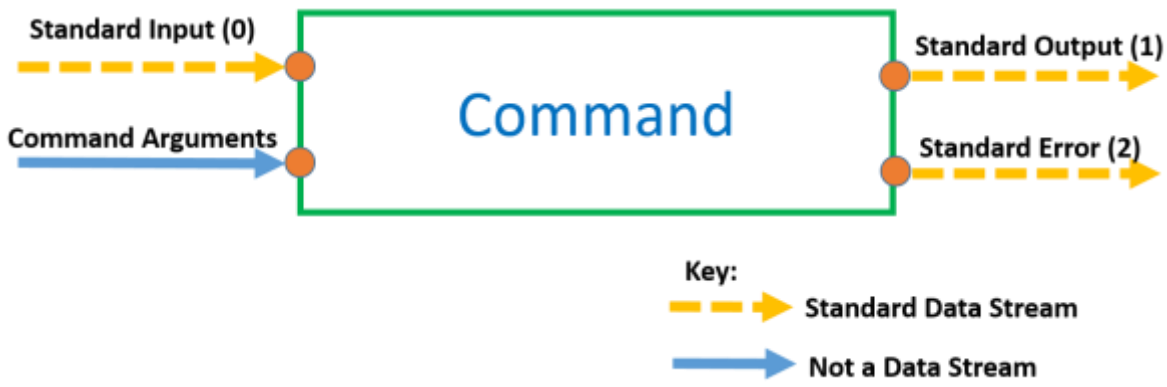
And the cal command has 2 command line arguments (12 and 2017).

6. Using the Manual

man –k <search term>	Search the manual for pages matching <search term>.
man 5 <page name>	Open the man page called <page name> in section 5 of the manual. (replace <page name> and 5 as required)
man <page name>	Open the man page called <page name> in section 1 of the manual.

See the Linux manual cheat sheet under the appropriate video for more information about the Linux manual and man pages.

7. Command Input and Output



Standard Data Streams can be redirected and are identified using their stream number.

Redirection of the standard output of one command to the standard input of another command is known as piping.

7.1 Redirecting Standard Output:

Standard output is stream number 1. There are 2 methods to redirect standard output.

The *long form*, using the stream number:

```
commandName –options arguments 1> destination
```

Or the *short form*, with no stream number:

```
commandName –options arguments > destination
```

7.2 Redirecting Standard Error:

Standard error is stream number 2.

Here is how to redirect standard error

```
commandName –options arguments 2> destination
```

Standard error can be redirected at the same time as standard output:

```
commandName –options arguments 1> output_destination 2> error_destination
```

7.3 Redirecting Standard Input:

Standard Input is stream number 0. There are 2 methods to redirect standard Input.

The *long form*, using the stream number:

```
commandName –options arguments 0< input_source
```

Or the *short form*, with no stream number:

```
commandName –options arguments < input_source
```

8. Piping

Piping is the connection of the standard output of one command to the standard input of another command. Piping using the pipe character (|) which is accessed by pressing SHIFT + BACKSLASH (\) on most keyboards.

Here is how you would pipe together commandOne and commandTwo:

```
commandOne –options arguments | commandTwo –options arguments
```

Notice how both commands can have their own options and command line arguments as usual. This piping can go on for as long as is required with as many commands as is required.

8.1 Taking "Snapshots" of pipeline data using the tee command

Redirecting during a pipeline breaks the pipeline.

For example, this **wouldn't** work:

```
commandOne –options arguments > snapshot.txt | commandTwo –options arguments
```

Because redirection is processed by the shell before piping is, snapshot.txt would be created, but this locks up the standard output stream and therefore no data can be passed through the pipeline to commandTwo.

NB: Redirection breaks pipelines

However, the tee command allows us to take a "snapshot" of the data in the pipeline **without** breaking the pipeline.

```
commandOne –options arguments | tee snapshot.txt | commandTwo –options arguments
```

Here, a snapshot of the data coming out of commandOne is saved in snapshot.txt, but the data is also successfully piped through to commandTwo.

8.2 Piping to commands that only accept command line arguments by using xargs

Piping connects the standard output of one command to the standard input of another command.

But what if the second command doesn't accept standard input? e.g. the echo command.

The key is to transform the data coming in, into command line arguments.

This is possible using the xargs command.

For example, this would **not** work:

```
commandOne –options arguments | echo
```

This **would** work:

```
commandOne –options arguments | xargs echo
```

9. Aliases

Aliases allow you to save your pipelines and commands with easy to remember nicknames so that they can be used later much easier.

You define aliases in your **.bash_aliases** file in your home directory. If it does not exist, you need to create it spelled exactly as shown. Note that the preceding period (.) must be included and there should be no file extension (such as .txt, or .pdf).

Here is how you define an alias in **.bash_aliases**:

alias aliasName="THING YOU WANT TO ALIAS"

Notice that there are no spaces between the equals sign (=) and the aliasName and the quotes (“). The quotes can be double quote (“) or single quotes (‘).

Let’s take an example:

```
alias calmagic="cal –A 1 –B 1 12 2017"
```

With this alias defined in our **.bash_aliases** file, whenever we run the calmagic command it is as if we ran the cal –A 1 –B 1 12 2017 command.

calmagic is now said to be an alias of “cal –A 1 –B 1 12 2017”.

NB: Aliases may contain either one command or an entire pipeline!

9.1 Piping to an alias

If the first command in an alias accepts standard input, then the alias can be piped to; even if it is an entire pipeline!

Our alias is currently:

```
alias calmagic="cal –A 1 –B 1 12 2017"
```

cal is the first command in this alias, but cal doesn’t accept standard input.

Therefore, this would **not** work:

```
commandOne –options arguments | calmagic
```

However, if we adjust our alias so that it *can* accept standard input.

```
alias calmagic="xargs cal –A 1 –B 1 12 2017"
```

This will now work:

```
commandOne –options arguments | calmagic
```

And yes, you can pipe out of an alias as well, if the alias produces standard output.

```
commandOne –options arguments | calmagic | commandTwo –options arguments
```

Think of aliases as building blocks that you can use in more sophisticated pipelines.

“Well done! You have learned an incredible amount of stuff in this section of the course and I hope that this cheat sheet is useful to you!

Best wishes, Ziyad.