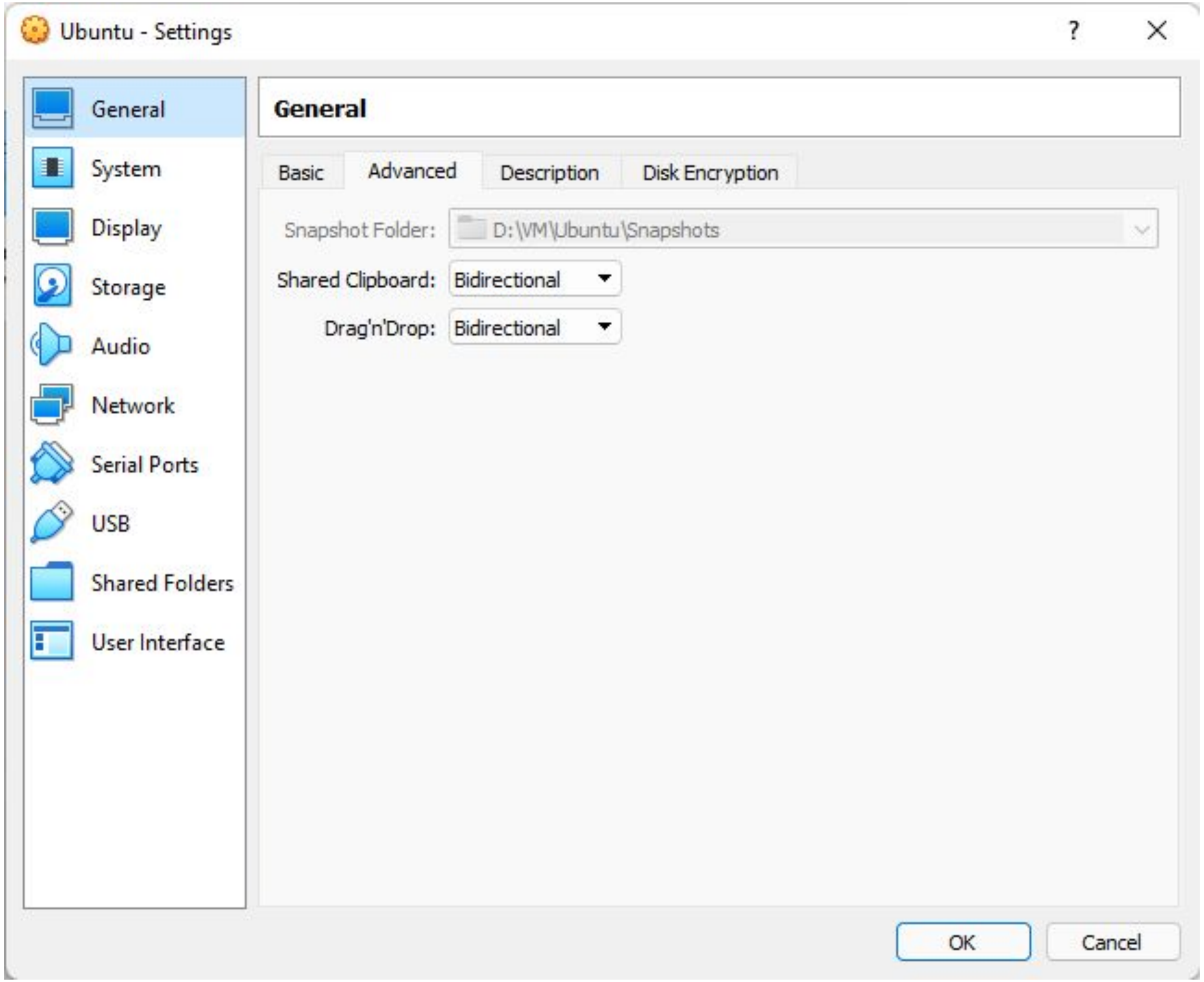
Quick review and new materials

CLARUSWAY
WAY TO REINVENT YOURSELF



Snapshot Folder: D:\VM\Ubuntu\Snapshots

Shared Clipboard: Bidirectional

Drag'n'Drop: Bidirectional

CLARUSWAY
WAY TO REINVENT YOURSELF

Quick review

cd / changes directory to physical root
mkdir xxx creates a directory
rm -r xxx directory recursively removes the file/folder
lspci, lshw, lsusb
pwd Displays the working directory
!! re-execute the latest command, use up/down keys
tabKey Using tab key finds the alternatives
touch creates a file.
cp copies a file with default or updated name
mv moves a file
cat display contents of a file
echo mainly used in scripts, also displays environment variables
.sh is an executable script file
dmesg investigate the startup logs
cat /proc/cpuinfo : details of CPU
/proc folder contains useful information
free -h or free : details of ram usage
nano and vi are the text editors
ip a : displays the IP configuration
env : displays the environment variables
chmod 777 file : assign the access permissions
ping xxx : check if there is a problem with physical access
ssh : open up a secure shell connection
host : host www.sybex.com

CLARUSWAY
WAY TO REINVENT YOURSELF

What defines a Linux OS?

- Command line shells
- Graphical User Interfaces (Ease of use for command line shells)
- Utility Programs (Calculators, disk managers etc)
- Libraries (Collections of functions)
- Productivity Programs (web browsers, word processors, graphic editors)

Optional : Install Blender in Ubuntu

- sudo apt-get update
- sudo apt-get install Blender
- blender

Installation can take around 10 min with ~900Mb space

Using tar archive extractor

- tar –help

- Create a new archive file
  tar -cf backup.tar /home/volkan

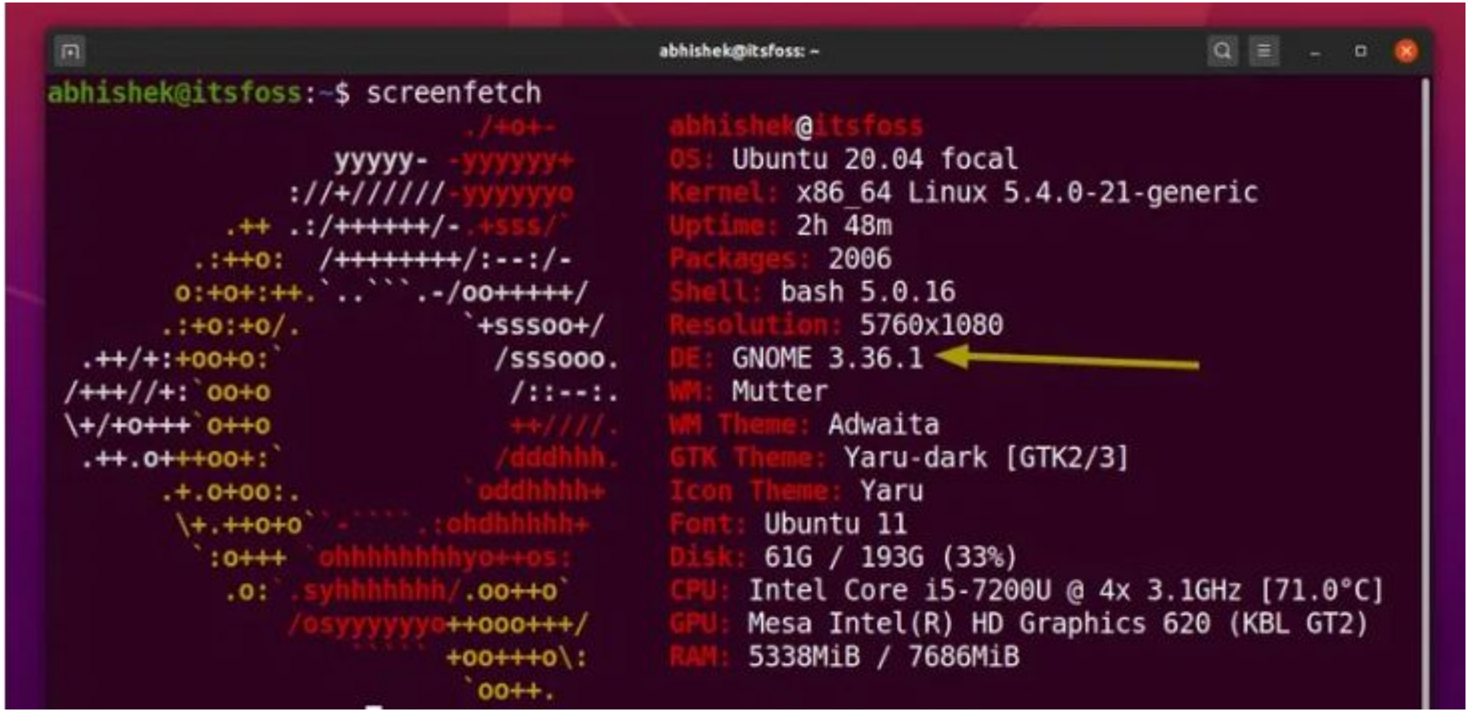
- tar -C /home/volkan/installers -xvf backup.tar

  Its also possible to use zip/unzip

Choosing a desktop environment

KDE
GNOME
LXDE
Unity
Xfce
Build Your Own

Choosing a desktop environment



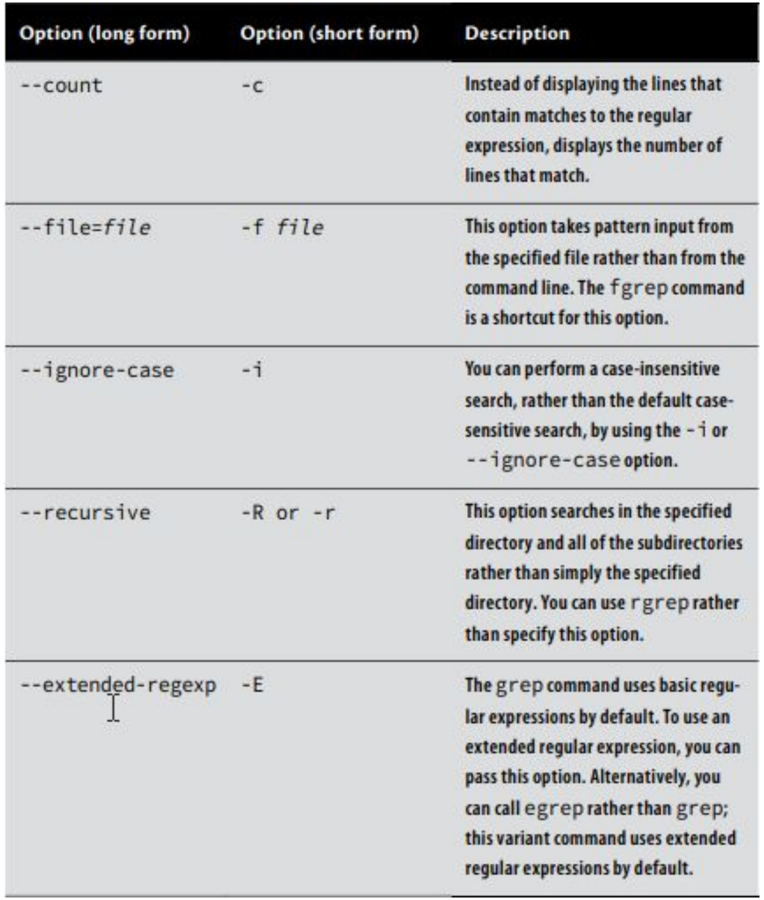
~~~
abhishek@itsfoss:~$ screenfetch
                          ./+o+-       abhishek@itsfoss
                  yyyyy- -yyyyyy+      OS: Ubuntu 20.04 focal
               ://+//////-yyyyyyo      Kernel: x86_64 Linux 5.4.0-21-generic
           .++ .:/++++++/-.+sss/`      Uptime: 2h 48m
         .:++o:  /++++++++/:--:/-      Packages: 2006
        o:+o+:++.`..```.-/oo+++++/     Shell: bash 5.0.16
       .:+o:+o/.          `+sssoo+/    Resolution: 5760x1080
  .++/+:+oo+o:`             /sssooo.   DE: GNOME 3.36.1
 /+++//+:`oo+o               /::--:.   WM: Mutter
 \+/+o+++`o++o               ++////.   WM Theme: Adwaita
  .++.o+++oo+:`             /dddhhh.   GTK Theme: Yaru-dark [GTK2/3]
       .+.o+oo:.          `oddhhhh+    Icon Theme: Yaru
        \+.++o+o``-````.:ohdhhhhh+     Font: Ubuntu 11
         `:o+++ `ohhhhhhhhyo++os:      Disk: 61G / 193G (33%)
           .o:`.syhhhhhhh/.oo++o`      CPU: Intel Core i5-7200U @ 4x 3.1GHz [71.0°C]
               /osyyyyyyo++ooo+++/     GPU: Mesa Intel(R) HD Graphics 620 (KBL GT2)
                   ````` +oo+++o\:     RAM: 5338MiB / 7686MiB
                            `oo++.
~~~

Common used package tools

dpkg
rpm
apt-get
yum

Using grep command

The grep command searches for files that contain a specified string and returns the name of the file and (if it's a text file) the line containing that string.
You can also use grep to search a specified file for a specified string.
To use grep , you type the command's name, an optional set of options, a regular expression, and an optional filename specification.
The grep command supports a large number of options



Option (long form)	Option (short form)	Description
`--count`	`-c`	Instead of displaying the lines that contain matches to the regular expression, displays the number of lines that match.
`--file=`*`file`*	`-f` *`file`*	This option takes pattern input from the specified file rather than from the command line. The `fgrep` command is a shortcut for this option.
`--ignore-case`	`-i`	You can perform a case-insensitive search, rather than the default case-sensitive search, by using the `-i` or `--ignore-case` option.
`--recursive`	`-R` or `-r`	This option searches in the specified directory and all of the subdirectories rather than simply the specified directory. You can use `rgrep` rather than specify this option.
`--extended-regexp`	`-E`	The `grep` command uses basic regular expressions by default. To use an extended regular expression, you can pass this option. Alternatively, you can call `egrep` rather than `grep`; this variant command uses extended regular expressions by default.

CLARUSWAY
WAY TO REINVENT YOURSELF

Using grep command

```
sudo grep -r eth0 /etc/*
```

This example finds all of the files in /etc that contain the string eth0
(the identifier for the first Ethernet device on most distributions).
Because the example includes the -r option, it searches recursively, so grep searches files in subdirectories of /etc as well as those in /etc itself.
For each matching text file, the line that contains the string is printed.

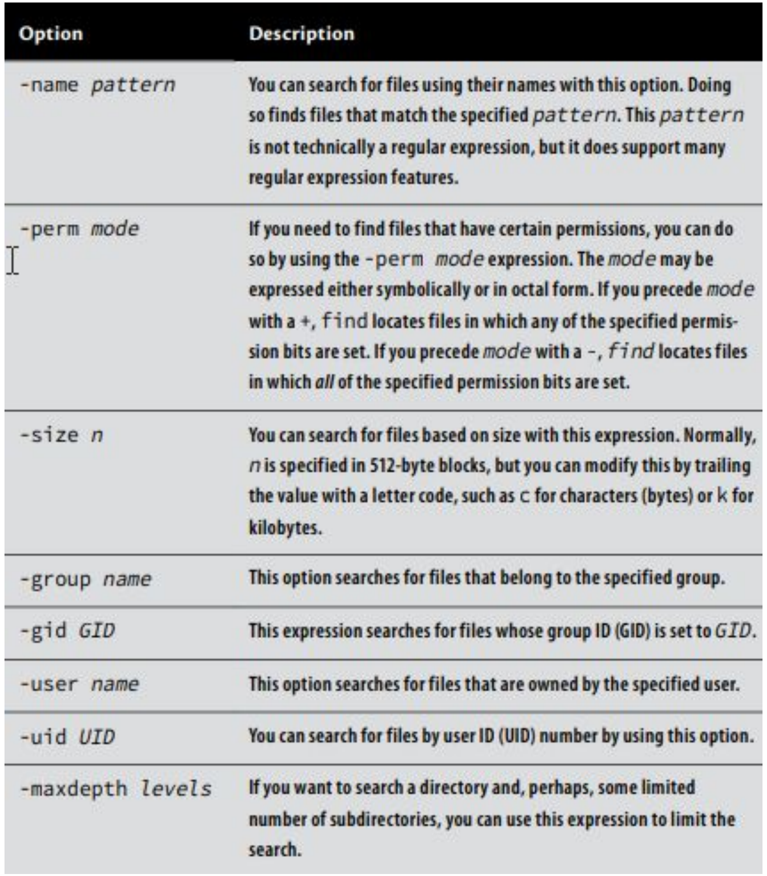
Suppose that you want to locate all of the files in /etc that contain the string eth0 or eth1 .
You can enter the following command, which uses a bracket regular expression to specify both variant devices:

```
sudo grep eth[01] /etc/*
```

Using find command

This utility application finds files by searching through the specified directory tree, checking filenames, file creation dates, and so on to locate the files that match the specified criteria.
Because of this operation method, find tends to be slow. It's flexible, however, and likely to succeed, assuming that the file for which you're searching exists.

sudo find /home -name "*.c"



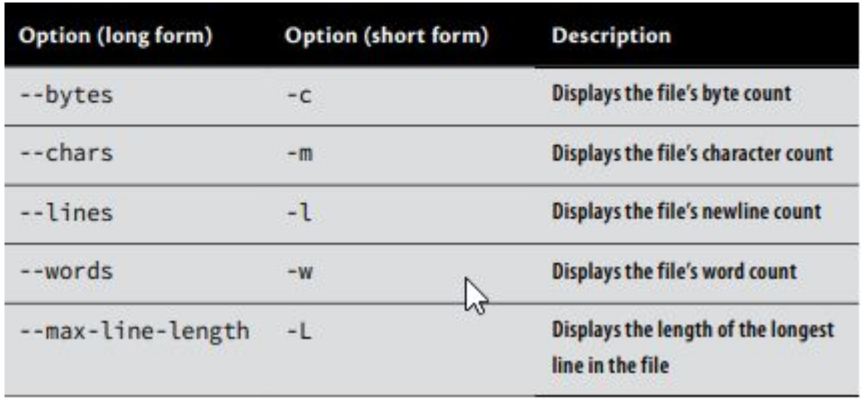
Option	Description
`-name pattern`	You can search for files using their names with this option. Doing so finds files that match the specified *pattern*. This *pattern* is not technically a regular expression, but it does support many regular expression features.
`-perm mode`	If you need to find files that have certain permissions, you can do so by using the `-perm mode` expression. The *mode* may be expressed either symbolically or in octal form. If you precede *mode* with a +, `find` locates files in which any of the specified permission bits are set. If you precede *mode* with a -, `find` locates files in which *all* of the specified permission bits are set.
`-size n`	You can search for files based on size with this expression. Normally, *n* is specified in 512-byte blocks, but you can modify this by trailing the value with a letter code, such as `c` for characters (bytes) or `k` for kilobytes.
`-group name`	This option searches for files that belong to the specified group.
`-gid GID`	This expression searches for files whose group ID (GID) is set to *GID*.
`-user name`	This option searches for files that are owned by the specified user.
`-uid UID`	You can search for files by user ID (UID) number by using this option.
`-maxdepth levels`	If you want to search a directory and, perhaps, some limited number of subdirectories, you can use this expression to limit the search.

CLARUSWAY
WAY TO REINVENT YOURSELF

Using wc command

A file’s size in bytes, as revealed by ls or searched for using find , can be a useful metric. This size value isn’t always the most useful one for text files, though. You might need to know how many words or lines are in a text file—say, because you want to know how many pages a text document will consume when printed at 52 lines per page.
The wc utility provides this information. For instance, to discover the information for a newly created file named newfile .txt in your present working directory:

```
$ wc newfile.txt 37 59 1990 newfile.txt
```



Option (long form)	Option (short form)	Description
`--bytes`	`-c`	Displays the file’s byte count
`--chars`	`-m`	Displays the file’s character count
`--lines`	`-l`	Displays the file’s newline count
`--words`	`-w`	Displays the file’s word count
`--max-line-length`	`-L`	Displays the length of the longest line in the file

Using sort command

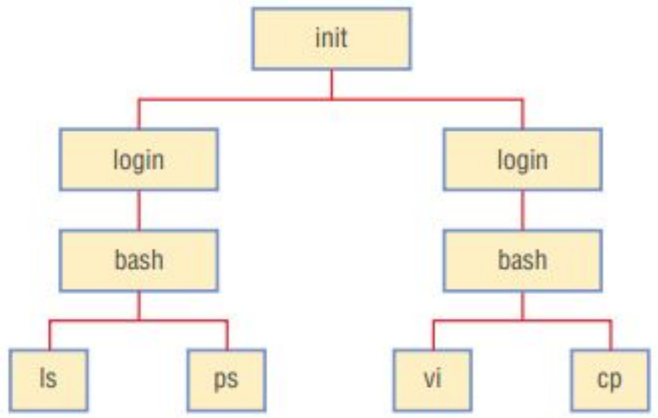
When dealing with a large amount of data, being able to sort it is often useful. The sort command does just that. However, you need to be aware of its features in order to achieve the desired results.

Please create a file named pets.txt and sort the internal elements by sort.

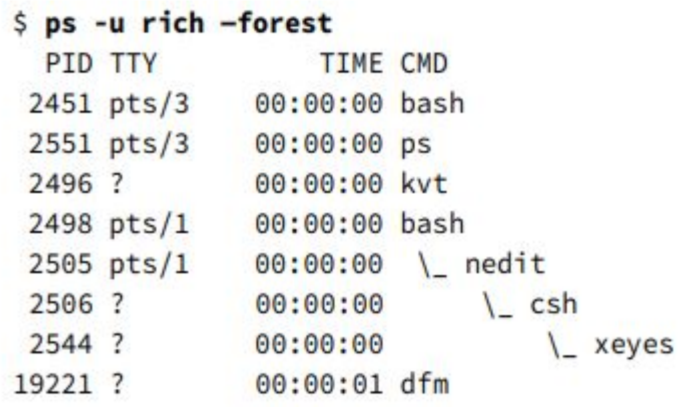
```
$ cat pets.txt
fish
dog
cat
bird
$ sort pets.txt
bird
cat
dog
fish
```

Option (long form)	Option (short form)	Description
`--dictionary-order`	`-d`	Considers only blanks and alpha-numeric characters; doesn't consider special characters
`--ignore-case`	`-f`	Ignores case (default is to consider case and order capitalized letters first)
`--numeric-sort`	`-n`	Sorts by string numeric value
`--output=`*`file`*	`-o`	Writes results to file specified
`--reverse`	`-r`	Sorts in descending order (default is to sort ascending)

Linux Processes



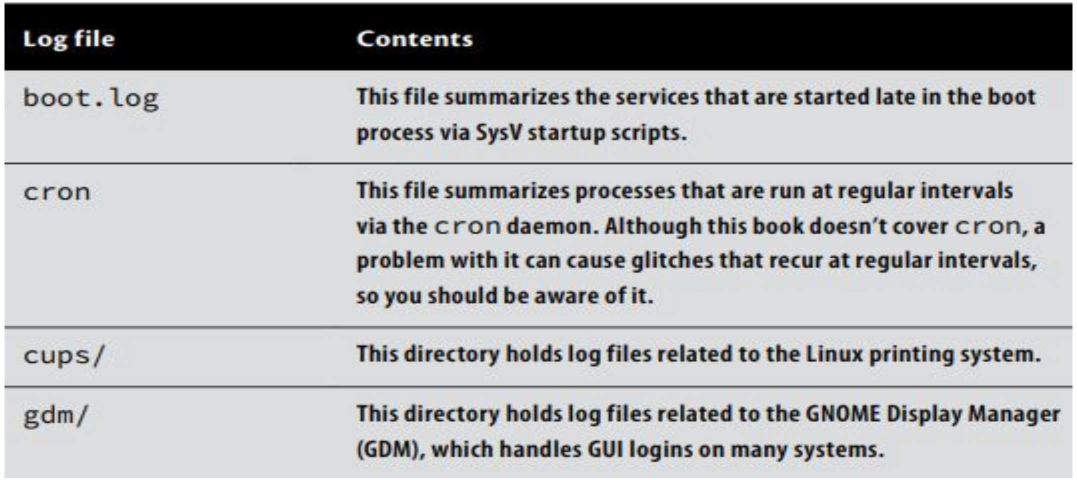
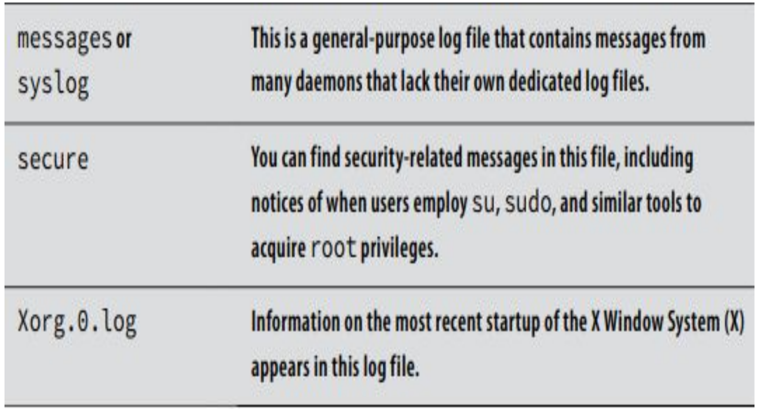
Before we can manage processes, we must be able to identify them. The **ps and top, bashtop** utilities can help you identify processes. In either case, you can search for processes in various ways, such as by name or by resource use. You may also want to identify how much memory your processes are consuming, which you can do with the free command.



```
$ ps -u rich -forest
  PID TTY          TIME CMD
 2451 pts/3    00:00:00 bash
 2551 pts/3    00:00:00 ps
 2496 ?        00:00:00 kvt
 2498 pts/1    00:00:00 bash
 2505 pts/1    00:00:00  \_ nedit
 2506 ?        00:00:00      \_ csh
 2544 ?        00:00:00          \_ xeyes
19221 ?        00:00:01 dfm
```

Using Log files

Linux stores most log files in the /var/log directory tree. Table 9.2 summarizes some common log files on many Linux systems. In addition, many server programs not described in this book add their own log files or subdirectories of /var/log . If you experience problems with such a server, checking its log files can be a good place to start troubleshooting.

Log file	Contents
`boot.log`	This file summarizes the services that are started late in the boot process via SysV startup scripts.
`cron`	This file summarizes processes that are run at regular intervals via the `cron` daemon. Although this book doesn't cover `cron`, a problem with it can cause glitches that recur at regular intervals, so you should be aware of it.
`cups/`	This directory holds log files related to the Linux printing system.
`gdm/`	This directory holds log files related to the GNOME Display Manager (GDM), which handles GUI logins on many systems.
`messages` or `syslog`	This is a general-purpose log file that contains messages from many daemons that lack their own dedicated log files.
`secure`	You can find security-related messages in this file, including notices of when users employ `su`, `sudo`, and similar tools to acquire `root` privileges.
`Xorg.0.log`	Information on the most recent startup of the X Window System (X) appears in this log file.

Creating scripts and using them in Linux

A script is a program written in an interpreted language, typically associated with a shell or a compiled program.
In Linux, many scripts are shell scripts , which are associated with Bash or another shell. (If you’re familiar with batch files in Windows, scripts serve a similar purpose.)

You can write shell scripts to help automate tedious, repetitive tasks or to perform new and complex tasks.
Scripts perform many of Linux’s startup functions, so mastering scripting will help you manage the startup process.
We will check the Bash shell scripts, beginning with the task of creating a new script file.
We then describe several important scripting features that help you to perform progressively more complex scripting tasks

The scripts begin with

```
#!/bin/bash
chmod a+x my-script
```

Creating scripts and using them in Linux

Echo and Printf commands

```
#!/bin/bash
echo This is a demonstration of a simple script
```

```
#!/bin/bash
printf “This is a demonstration of a simple script\n”
```

Creating scripts and using them in Linux

printf FORMAT [ARGUMENTS]

```
#!/bin/bash
printf "%d mul %f = %f\n" 6 6.0 36.0
```

The output would be :

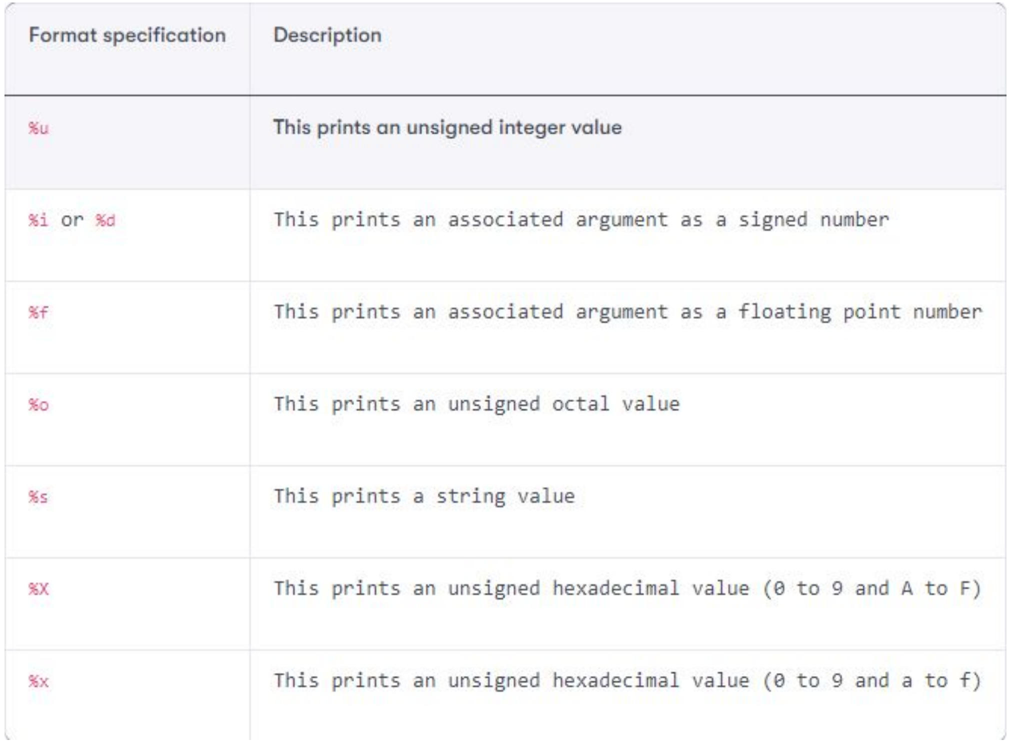
```
6 mul 6.000000 = 36.000000
```

Another example :

```
printf "%d mul %.2f = %.2f\n" 6 6.0 36.0
```

The output would be :

```
6 mul 6.00 = 36.00
```

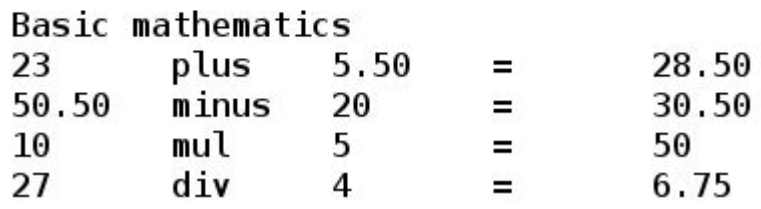


Format specification	Description
`%u`	This prints an unsigned integer value
`%i` or `%d`	This prints an associated argument as a signed number
`%f`	This prints an associated argument as a floating point number
`%o`	This prints an unsigned octal value
`%s`	This prints a string value
`%X`	This prints an unsigned hexadecimal value (0 to 9 and A to F)
`%x`	This prints an unsigned hexadecimal value (0 to 9 and a to f)

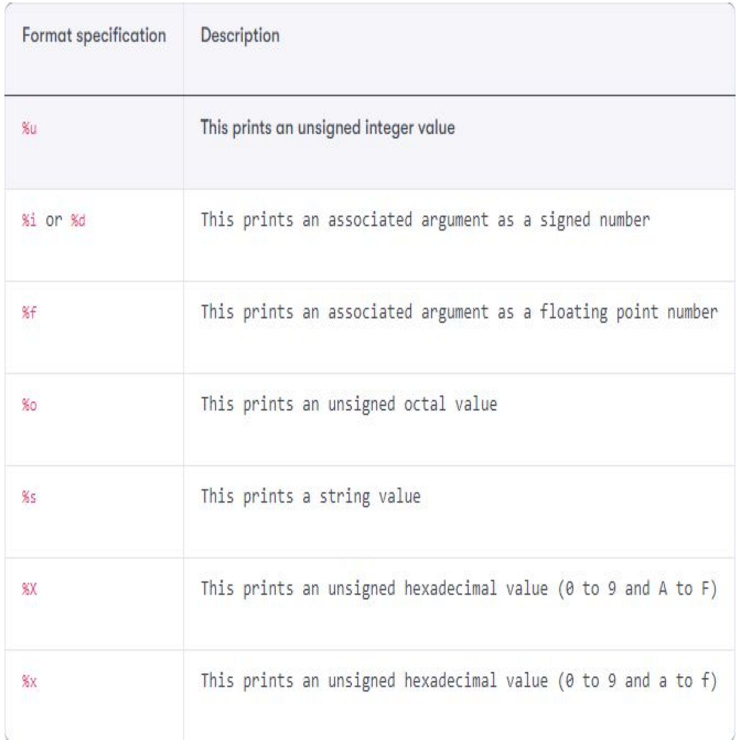
Creating scripts and using them in Linux

```
#!/bin/bash
#Filename: print.sh
#Description: print and echo

echo "Basic mathematics"
printf "%-7d %-7s %-7.2f =\t%-7.2f\n" 23 plus 5.5 28.5
printf "%-7.2f %-7s %-7d =\t%-7.2f\n" 50.50 minus 20 30.50
printf "%-7d %-7s %-7d =\t%-7d\n" 10 mul 5 50
printf "%-7d %-7s %-7d =\t%-7.2f\n" 27 div 4 6.75
```

The output would be :



```
Basic mathematics
23      plus    5.50    =       28.50
50.50   minus   20      =       30.50
10      mul     5       =       50
27      div     4       =       6.75
```



Format specification	Description
%u	This prints an unsigned integer value
%i or %d	This prints an associated argument as a signed number
%f	This prints an associated argument as a floating point number
%o	This prints an unsigned octal value
%s	This prints a string value
%X	This prints an unsigned hexadecimal value (0 to 9 and A to F)
%x	This prints an unsigned hexadecimal value (0 to 9 and a to f)

Creating scripts and using them in Linux

A simple script that starts 3 applications :

```
#!/bin/bash
gnome-terminal &        // will work
/usr/bin/xterm &        // won’t work if not exist
/usr/bin/kmail &        // won’t work if not installed
```

The scripts may go complicated :

```
#!/bin/bash ip=`route -n | grep UG | tr -s “ “ | cut -f 2 -d “ “`
ping=”/bin/ping”
echo “Checking to see if $ip is up...”
$ping -c 5 $ip
```

Here the dollar sign indicates a variable

Creating scripts and using them in Linux

Using variables :

```
book="Linux Shell Scripting"  # Stores string value
book = "Linux Shell Scripting"  # Wrong, spaces around = operator
total_chapters=8    # Stores integer value
number_of_pages=210    # Stores integer value
average_pages_per_chapter=26.25    # Stores float value


echo average_pages_per_chapter
```

Creating scripts and using them in Linux

Accessing variables :

```
#!/bin/bash
#Filename: variables.sh
#Description: Basic variable definition and accessing them
echo Basic variable definition and accessing them

book="Linux Shell Scripting"
total_chapters=8
number_of_pages=210
average_pages_per_chapter=26.25

echo "Book name - $book"
echo "Number of Chapters - $total_chapters"
printf "Total number of pages in book - $number_of_pages\n"
printf "Average pages in each chapter - %-.2f\n" $average_pages_per_chapter
```

Creating scripts and using them in Linux

The unset script command

```
#!/bin/bash
#Filename: unset.sh
#Description: removing value of a variable

fruit="Apple"
quantity=6
echo "Fruit = $fruit , Quantity = $quantity"
unset fruit
echo "Fruit = $fruit , Quantity = $quantity"
```

The output would be :

```
Fruit = Apple , Quantity = 6
Fruit =  , Quantity = 6
```

CLARUSWAY
WAY TO REINVENT YOURSELF

Creating scripts and using them in Linux

Definition and usage of constant variables

```
#!/bin/bash
#Filename: constant.sh
#Description: constant variables in shell

readonly text="Welcome to Linux Shell Scripting"
echo $text
declare -r number=27
echo $number
text="Welcome"
```

The output would be :

```
Welcome to Linux Shell Scripting
27
constant.sh: line 9: text: readonly variable
```

CLARUSWAY
WAY TO REINVENT YOURSELF

Creating scripts and using them in Linux

Reading variables from user input and using them

```
#!/bin/bash
#Filename: inputs.sh
#Description: Reading the user input

read name loginname password
echo $name $loginname $password
```

The output would be : just check it on VBox :)

Creating scripts and using them in Linux

Reading variables from user input and using them

```
#!/bin/bash
#Filename: inputs.sh
#Description: Reading the user input


read -p "What is your name? "    # -p allows to prompt user a message
    What is your name? Foo
$  echo $REPLY
    Foo
```

The output would be as displayed above

Creating scripts and using them in Linux

Reading variables from user input and using them

```
#!/bin/bash
#Filename: inputs.sh
#Description: Reading the user input but in hidden form


read -s -p "Enter your secret key:"  # -s doesn't echo input in console
Enter your secret key:$    #Pressing enter key brings command prompt $
echo $REPLY
foo
```

The output would be as displayed above

Creating scripts and using them in Linux

A sample script that searches a file

```
#!/bin/bash
#Filename: read.sh
#Description: Find a file in a path entered by user

read -p "Enter filename to be searched:"
filename=$REPLY
read -p "Enter path for search:" path
echo "File $filename search matches to"
find $path -name $filename
```

The output would be as:

```
Enter filename to be searched:read
Enter path for search:/usr/bin
File read search matches to
/usr/bin/read
```

Creating scripts and using them in Linux

The if / fi usage

```
#!/bin/sh

a=10
b=20

if [ $a == $b ]
then
   echo "a is equal to b"
else
   echo "a is not equal to b"
fi
```

The output would be as:

```
a is not equal to b
```

CLARUSWAY
WAY TO REINVENT YOURSELF

Creating scripts and using them in Linux

The while do done

```
#!/bin/bash
x=1
while [ $x -le 5 ]
do
  echo "Welcome $x times"
  x=$(( $x + 1 ))
done
```

The output would be on your local

Creating scripts and using them in Linux

The if / fi usage

```
#!/bin/sh

a=0
while [ "$a" -lt 10 ]    # this is loop1
do
   b="$a"
   while [ "$b" -ge 0 ]  # this is loop2
   do
      echo -n "$b "
      b=`expr $b - 1`
   done
   echo
   a=`expr $a + 1`
done
```

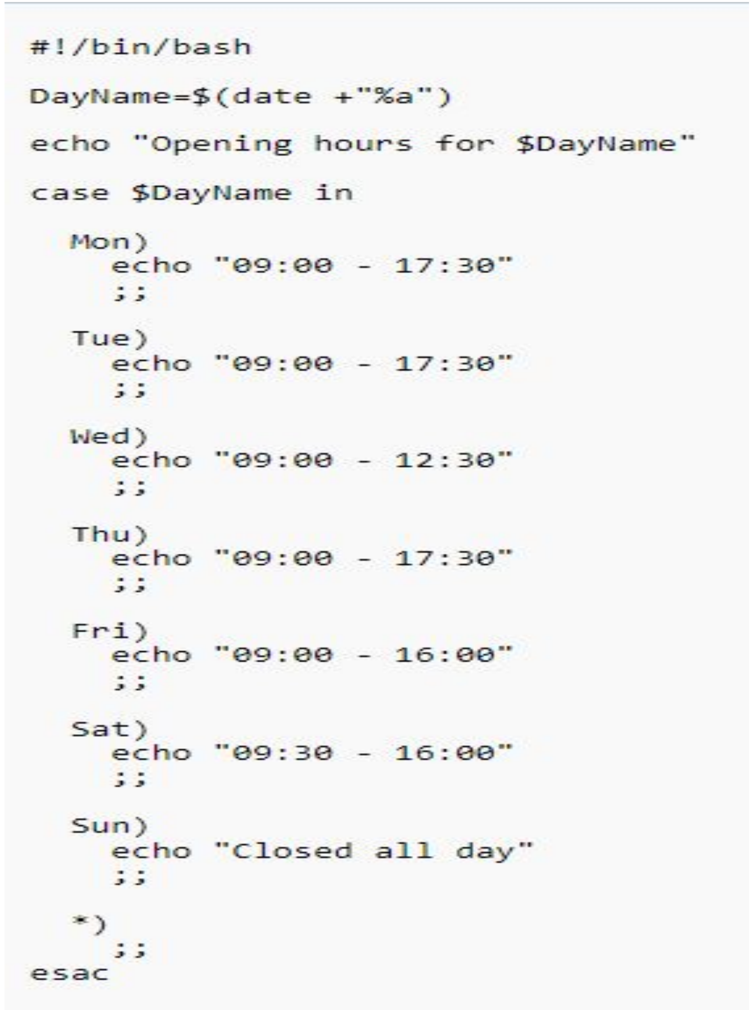
The output would be as:

```
0
1 0
2 1 0
3 2 1 0
4 3 2 1 0
5 4 3 2 1 0
6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
```

Creating scripts

```
if [ -s /tmp/tempstuff ]
        then echo "/tmp/tempstuff found; aborting!"
exit fi
```

```
case word in pattern1)

command(s);;
pattern2)
command(s);; ...

esac
```



```
#!/bin/bash

DayName=$(date +"%a")

echo "Opening hours for $DayName"

case $DayName in

  Mon)
    echo "09:00 - 17:30"
    ;;

  Tue)
    echo "09:00 - 17:30"
    ;;

  Wed)
    echo "09:00 - 12:30"
    ;;

  Thu)
    echo "09:00 - 17:30"
    ;;

  Fri)
    echo "09:00 - 16:00"
    ;;

  Sat)
    echo "09:30 - 16:00"
    ;;

  Sun)
    echo "Closed all day"
    ;;

  *)
    ;;
esac
```

CLARUSWAY
WAY TO REINVENT YOURSELF

Creating scripts

How to define functions in scripts

```
#!/bin/sh

# Define your function here
Hello () {
   echo "Hello World"
}

# Invoke your function
Hello
```

The output would be

```
$./test.sh
Hello World
```

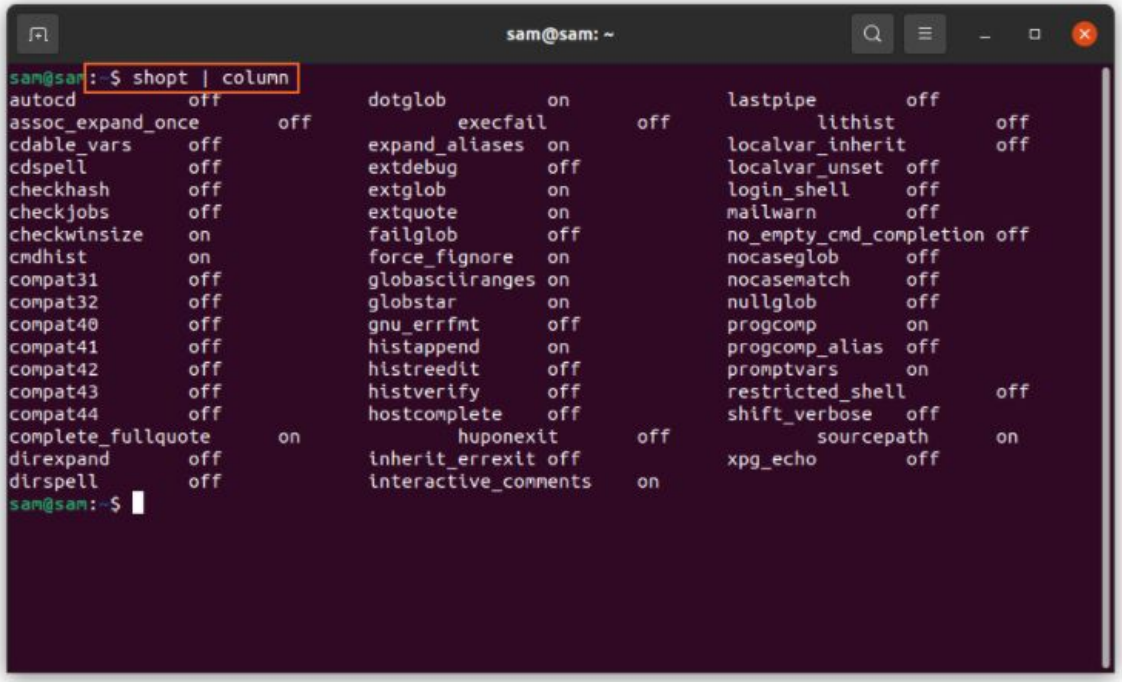
Creating scripts

```
#!/bin/bash

shopt -s nocasematch

echo "Enter name of a month"
read month

case $month in

  February)
    echo "28/29 days in $month"
    ;;

  April | June | September | November)
    echo "30 days in $month"
    ;;

  January | March | May | July | August | October | December)
    echo "31 days in $month"
    ;;

  *)
    echo "Unknown month: $month"
    ;;
esac
```

CLARUSWAY
WAY TO REINVENT YOURSELF

Creating scripts

```
$shopt | column
```



```
sam@sam:~$ shopt | column
autocd          off             dotglob         on              lastpipe        off
assoc_expand_once       off             execfail        off             lithist         off
cdable_vars     off             expand_aliases  on              localvar_inherit        off
cdspell         off             extdebug        off             localvar_unset  off
checkhash       off             extglob         on              login_shell     off
checkjobs       off             extquote        on              mailwarn        off
checkwinsize    on              failglob        off             no_empty_cmd_completion off
cmdhist         on              force_fignore   on              nocaseglob      off
compat31        off             globasciiranges on              nocasematch     off
compat32        off             globstar        on              nullglob        off
compat40        off             gnu_errfmt      off             progcomp        on
compat41        off             histappend      on              progcomp_alias  off
compat42        off             histreedit      off             promptvars      on
compat43        off             histverify      off             restricted_shell        off
compat44        off             hostcomplete    off             shift_verbose   off
complete_fullquote      on              huponexit       off             sourcepath      on
direxpand       off             inherit_errexit off             xpg_echo        off
dirspell        off             interactive_comments    on
sam@sam:~$
```

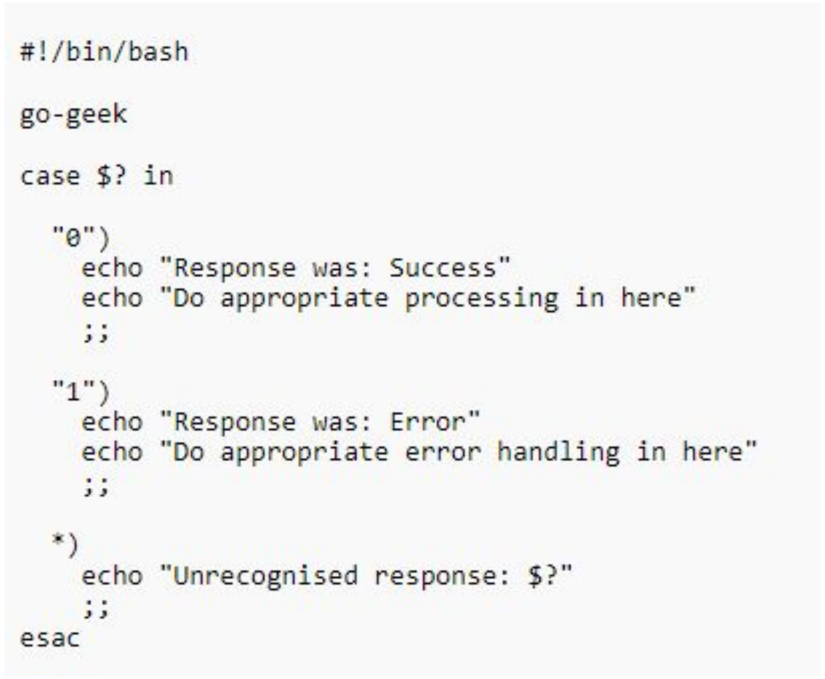
Creating scripts

```
#!/bin/bash

echo "Enter 1, 2, or 3: "
read Number

case $Number in

  "1")
    echo "Clause 1 matched"
    ;;

  "2")
    echo "Clause 2 matched"
    ;;

  "3")
    echo "Clause 3 matched"
    ;;

  *)
    echo "Default clause matched"
    ;;
esac
```

Using for statement in scripts

```bash
#!/bin/bash

for File in $(ls)

do
  # extract the file extension
  Extension=${File##*.}

  case "$Extension" in

    sh)
      echo " Shell script: $File"
      ;;

    md)
      echo " Markdown file: $File"
      ;;

    png)
      echo "PNG image file: $File"
      ;;

    *)
      echo "Unknown: $File"
      ;;
  esac
done
```

Using exit code in scripts



```
#!/bin/bash

go-geek

case $? in

  "0")
    echo "Response was: Success"
    echo "Do appropriate processing in here"
    ;;

  "1")
    echo "Response was: Error"
    echo "Do appropriate error handling in here"
    ;;

  *)
    echo "Unrecognised response: $?"
    ;;
esac
```

THANK YOU

Any questions?