Basics of the Linux terminal and tools

24/12/2021 By An7ar35

Contents

1 Nomenclature

All arguments in this document are surrounded with '`<`' and '`>`' signs to identify them as such. When typing out the arguments in the terminal make sure to replace the content including the signs with your own appropriate arguments.

<file name> A file name. e.g.: '`name_of_file.txt`' or '`~/Documents/name_of_file.txt`'.

<folder> A directory. e.g.: '`/home/dave1/Documents`' or '`Documents`'.

<path> A path to a directory or a file. e.g.: '`/home/dave1/Documents`' and '`/home/dave1/Documents/mydoc.txt`'.

<device> A device name. e.g.: '`/dev/sda1`'.

<username> A linux username. e.g.: '`bob123`'.

<group> A linux group name. e.g.: '`wheel`'.

<groups> List of linux group names. e.g.: '`wheel,sudo,audio`'.

<address> Network address or name. e.g.: '`192.168.1.1`' or '`www.duckduckgo.com`'.

<domain name> Domain name such as '`duckduckgo.com`'.

Options can generally be concatenated together when option specific arguments are not required. e.g.: `ls -l -s` can be written as `ls -ls` instead.

2 Introduction

This document outlines everything that one needs to get going in the Linux command line environment (Bash) and the wonderful tools that for and alongside it. It's meant to serve both as a reference guide and a general overview of what could be considered a basic working understanding of Linux from a command line perspective.

With consideration of the above, the command line utilities and their options highlighted here only cover a section of use cases and, thus, should be considered far from comprehensive. These just serve as a quick reference to get through some of the most common use cases. To get a full picture of all the options available for each commands check chapter 2.1 below.

2.1 Getting help

There are a couple of ways to get some indications as to what a command line utility can do and what arguments (options) can be passed to it. Normally both approach below should work but be aware that some less than standard command line tools may not have a `man` page implemented (it's rare but it happens).

Getting help

$ `<command> --help` : Getting a brief overview of the available options for a command.

$ `man <command>` : Internal **man**ual page for the command (use '`q`' to exit).

If things are still confusing don't underestimate a quick web search to find a solution. The GNU manual page for core utilities is a good place to start for a full list of the core commands available on Linux.

For all GNU Bash related information check out the official Bash Reference Manual.

3 Useful Concepts and Tooling

3.1 Command history

Bash retains a history (`~/.bash_history`) of the commands typed inside the user's home directory. Too see it in the terminal just type:

```
$ history
```

Note: The ↑ and ↓ keys can also be used to browse the past command line history line by line.

To avoid adding a command to this history just add a space at the beginning of the line for that command inside the terminal.

To do a reverse history search of a command use: Ctrl + R Repeating this will cycle through the alternatives found.

HSTR (optional)

Another option to browse/search through the history is to install HSTR. It's a supercharged and nicer to use history lookup replacing the traditional reverse-history search.

3.2 Command Redo/Undo/Edit

cmd	details
`fc`	Opens up an editor with the last command entered so that it can be fixed up.
Ctrl + X then Ctrl + E	Opens up an editor with the last command entered so that it can be fixed up.
`!!`	Takes the last command in the history.

For example to run the last command typed as root:

```
$ sudo !!
```

3.3 Piping ('|')

A pipe (`|`) redirects a program's output to another program for further processing. For example: getting a list of all the files in a directory (`ls`) and then filtering (`grep`) the resulting output for any entries that have 'host' in their names. I.e.:

```
$ ls | grep 'host'
```

Multiple application can be chained up as such:

```
command_1 | command_2 | command_3 ...
```

For example: getting a list of files that have 'host' in their name and, out of that result, get only the files that also have 'name' in their names:

```
$ ls | grep 'host' | grep 'name'
```

Another example: see all messages that include the string 'error' (`grep -i 'error'` - case insensitive) produced by linux on startup (`dmesg`) in a scrollable application (`less`):

```
$ dmesg | grep -i 'error' | less
```

3.4 Chaining ('&&')

In order to chain commands so that they execute one after the other the double ampersand `&&` signs can be used to separate each commands in the sequence.

To note that `&&` only executes if the previous command had an exit status of `0` (i.e.: finished without failing).

For example: show all files in the current directory and then print the message '`All done!`' afterwards.

```
$ ls && echo "All done!"
```

4 Basic Operations

4.1 Files and Directories

cmd	opt	args	details
`touch`		*<file name>*	create blank file
`>`		*<file name>*	create blank file
`rm`		*<file name>*	**rem**ove file
`rm`	`-r`	*<path>*	**rem**ove all files and folders recursively in path
`cd`		*<path>*	**c**hange **d**irectory
`cd`		`..`	go up 1 directory
`cd`		`../..`	go up 2 directories
`cd`		`/`	go to root directory
`cd`		`~`	go to current user's home directory
`pwd`			Print the full filename of the current working directory.
`pwd`	`-L`		Print the full Logical filename of the current working directory.
`pwd`	`-P`		Print the full Physical filename of the current working directory.
`mkdir`		*<folder>*	**m**a**k**e **dir**ectory
`mkdir`	`-p`	*<folder>*	**m**a**k**e **dir**ectory and create parent directories as needed
`mkdir`	`-m`	*<mode> <folder>*	**m**a**k**e **dir**ectory and set its permission (octal mode - see 8.5)
`rmdir`		*<folder>*	**rem**ove an empty **dir**ectory
`rmdir`	`-p`	*<path>*	**rem**ove an empty **dir**ectory and empty parent(s) in path

4.2 Getting information

cmd	opt	details
`ls`		**lis**t folders and files in current directory
`ls`	`-d`	**lis**ts all directories in current directory
`ls`	`-l`	**lis**ts folders and files in current directory in long format
`ls`	`-hs`	**lis**ts folders and files in current directory with human readable sizes
`dir`		lists all **dir**ectories in current directory

4.3 Copying, Moving and Renaming

cmd	args	details
`cp`	*<source> <target>*	**copy** a folder/file
`mv`	*<source> <target>*	**move**/rename a folder/file

Example

```
cp source_path/file_name.txt target_path/file_name.txt
```

4.4 Searching

The `find` utility is pretty straight forward but can also be powerful. Note that there are many options available (see `man` page) beyond just searching for a file name (e.g.: last modified, last used, size, user/group ownership, etc...).

cmd	opt/args	details
`find`	`-name` *<file name>*	Search for a file name from the current location.
`find`	*path* `-name` *<file name>*	Search for a file name from a given location.

Examples

```
//Search for files with the 'txt' extension from the home directory
find ~ -name *.txt

//Search for files name log from the root directory irrespective of the extension
find / -name log.*
```

4.5 Comparison

The `diff` command can compare text based files as well as directories (replace *file1* and *file2* with folder paths).

cmd	opt	args	details
`diff`		*<file1> <file2>*	Shows differences between 2 text files.
`diff`	`-s`	*<file1> <file2>*	Verifies if the files are identical.
`diff`	`-q`	*<file1> <file2>*	Verifies if the files are **not** identical.
`diff`	`-y`	*<file1> <file2>*	Show file comparison side-by-side.
`diff`	`--suppress-common-lines`	*<file1> <file2>*	Show only changed/added/deleted lines.
`diff`	`-b`	*<file1> <file2>*	Verifies if the files are identical but ignore any changes which only change the amount of white-space (spaces/tabs).
`diff`	`-Z`	*<file1> <file2>*	Verifies if the files are identical but ignore any trailing white-space.
`diff`	`-w`	*<file1> <file2>*	Verifies if the files are identical but ignore white-space entirely.
`diff`	`-i`	*<file1> <file2>*	Case-insensitively verifies if the files are identical.

When running `diff` between two files, the output always describes what changes would be needed to transform the first into the second so that both match.

In single column view, the way `diff` show that is by by giving the line numbers from the first file, the action character (see right), then the line numbers from the second file.

e.g.: `1,2c1,2` which means that line 1 through 2 in first file were changed to line 1 through 2 in the second file.

For directories, the output describes the files and folders differences.

Legend

Single column view:

`a`	add
`c`	change
`d`	delete
`<`	line from first file
`>`	line from second file
`---`	file separator

Side-by-side view:

`|`	line changed
`>`	line added
`<`	line deleted

Example 1: Text files

a.txt

```
My name is Bart Simpson.
I live in Springfield.
```

b.txt

```
My name is Lisa Simpson.
I live in Springfield.
I play the saxophone.
```

Result of `diff a.txt b.txt`

```
1c1
< My name is Bart Simpson.
---
> My name is Lisa Simpson.
2a3
> I play the saxophone.
```

So here line 1 in ‘`b.txt`’ was changed and the ”I play the saxophone.” on line 3 was added from the perspective of the ‘`a.txt`’ file.

Note

If you find yourself working on different versions of text-based files you might want to look into a Version Control System such as GIT or Subversion (SVN).

Other command line tools[1]

cmd	details
`colordiff`	Perl wrapper for `diff` that provides some colour and syntax highlighting as well as customisable colour schemes improving the overall experience.
`wdiff`	Front end for `diff` that provides the facility to compare on a word-by-word basis.

5 Text manipulation

Note

All text manipulation tools accept piped streamed input (’`|`’).

5.1 Filters

5.1.1 awk

AWK is a scripting language whose purpose is to manipulate data and generate reports. The `awk` command line tool uses the language but requires no compiling. It is most commonly used for pattern searching and matching in documents.

[1] Not installed by default.

As it is a rather specialised topic it falls outside the scope of this text. It is however a tool that anyone interested in data/text processing should be aware of and have a basic know-how of its uses at the very least.

cmd	opt	args	details
`awk`		*{<script>} <file name>*	Execute an AWK script on a file's data.

Examples

```
//Prints all lines from data.txt that contain 'Unix'.
awk '/Unix/' data.txt

//Prints lines 5 to 10 from data.txt.
awk 'NR>=5&&NR<=10' data.txt

//Prints the date (col 7) and month (col 6) for each files/folders.
ls -l | awk '{printf "%s %s \n", $7, $6}'
```

Some useful links:

- The AWK language
- GNU AWK Manual (`gawk`)
- Linux `awk` manual

5.1.2 grep

Global **r**egular **e**xpression **p**rint is used for matching regular expression against text in file(s) or a streamed input and outputting the resulting matches.

cmd	opt	args	details
`grep`		*<regex pattern> <file name>*	Find and output lines that have a match.
`grep`	`-n`	*<regex pattern> <file name>*	Find and output lines along with their line number(s) that have a match.
`grep`	`-i`	*<regex pattern> <file name>*	Find and output lines that have a case-insensitive match.
`grep`	`--color`	*<regex pattern> <file name>*	Find and output lines that have a match with the matched pattern in colour.

This tool can search multiple files (by use of a path + wildcard) as well as streamed inputs (using piping). It offers a multitude of other options making it a very powerful tool worth knowing about (see GNU `grep` manual).

5.1.3 sed

Stream **ed**itor's common usage case includes substitution, removal and, of course, filtering. Its uses can overlap with `awk`.

Here, we are just going to go over the substitution.

cmd	opt	args	details
`sed`		*<pattern> <file name>*	Apply the replacement pattern to file and print results.
`sed`	`-e`	*<patterns> <file name>*	Apply the replacement patterns to file and print results.
`sed`	`-n`	*<patterns> <file name>*	Apply the replacement patterns to file and print only modified lines.

Patterns are formatted as such: `s/pattern to match/replacement/flags`. There are 4 types of flags, and they are optional as the default is to match and replace the first occurrence on each lines..

1. `g`: (global) replace all occurrences,
2. *n*: the *n*th match on each line will be substituted,
3. `p`: print the original content,
4. `w <file>`: means write the results to a file.

For multiple patterns a `;` is used to separate them: `'s/pattern 1/replacement 1/flag; s/pattern 2/replacement 2/flag'`

Examples (using a fictional 'data.txt' as source)

Replacing text

```
//Replace all first found instances of 'Berlin' in each lines with 'London'
sed 's/Berlin/London/' data.txt

//Replace all first found instances of (a) 'Anna' with 'Celine' and (b) 'Bob' with
//'John Wick' in each lines.
sed -e 's/Anna/Celine/; s/Bob/John Wick/' data.txt

//Replace all found instances of 'the' with 'this' in line 3
sed '3s/the/this/g' data.txt

//Replace all found instances of 'the' with 'this' in lines 3 → 8
sed '3,8s/the/this/g' data.txt

//Replace all found instances of 'the' with 'this' in lines 3 → end of file
sed '3,$s/the/this/g' data.txt
```

Deleting lines text

```
//Delete the 2nd line
sed '2d' data.txt

//Delete lines 5 → 10
sed '5,10d' data.txt
```

Inserting or appending lines

```
//Insert line 'this is cool' before at the beginning of the text
sed 'i/this is cool' data.txt

//Append line 'this was cool' after text
sed 'a/this was cool' data.txt
```

5.2 Slicing and extracting

5.2.1 head and `tail`

When passing a file or piping a stream to either `head` or `tail` it takes a chunk of a specified size of just the beginning or end respectively. When dealing with files, the target file name is appended to the end of the arguments.

cmd	opt	args	details
`head`	`-c` / `--bytes=`	*<num>*	Prints the first `num` bytes of each file. When prefixed with a `-`; prints all but the last `num` bytes of each file. A multiplier suffix can be added: `b`, `kB`, `K`, `MB`, `M`, `GB` `G`, ...
`head`	`-n` / `--lines=`	*<num>*	Prints the first `num` lines (default=10). When prefixes with `-`; prints all but the last `num` lines of each file.

cmd	opt	args	details
`tail`	`-c` / `--bytes=`	*<num>*	Prints the last `num` bytes of each file. When prefixed with a `+`; prints all bytes from and including the byte `num`. A multiplier suffix can be added: `b`, `kB`, `K`, `MB`, `M`, `GB` `G`, ...
`tail`	`-n` / `--lines=`	*<num>*	Prints the last `num` lines (default=10). When prefixed with a `+`; prints all lines from and including line `num`.
`tail`	`-f` / `--follow`		Keeps an eye on the target file and prints whatever and whenever new data is added to the end of said file.
`tail`	`-F`		Same as `-f` but also retries to open a file even if temporarily inaccessible.
`tail`	`--pid=`	*<PID>*	Terminate operations when following a file (`-f`) with the given `PID` dies.

Example: Prints any new lines with "`error`" generated from a log

Here we are piping the output of `tail` into `grep` to filter just the updates we are interested in (the ones with "error" in them).

```
tail -f server.log | grep -i error
```

5.2.2 `cut`

The `cut` command is another extensive tool that, in simple terms, removes sections from each line of files. It can be extremely useful in extracting data from large sets and is worth learning about in more details.

opt	args	details
`-b`, `--bytes=`	`LIST`	Select only the listed bytes.
`-c`, `--characters=`	`LIST`	Select only the listed characters.
`-d`, `--delimiter=`	`DELIM`	Use `DELIM` instead of `TAB` as field delimiter.
`-f`, `--fields=`	`LIST`	Select only the listed fields; also print any line that contains no delimiter character, unless the `-s` option is specified.
`-n`		(ignored)
`--complement`		Complement the set of selected bytes, characters or fields.
`-s`, `--only-delimited`		Do not print lines not containing delimiters.
`--output-delimiter=`	`STR`	Use `STR` as the output delimiter (default: input delimiter).
`-z`, `--zero-terminated`		State that the line delimiter is NULL, not newline.

There can only be one exclusively of the other of the following options: `-b`, `-c` or `-f`.

Lists (`LIST`) are comprised of 1 or more ranges: `RANGE` or `RANGE_1,RANGE_2,...,RANGE_N`

Ranges are formatted as such:

`N`	: `N`'th byte, character or field, counted from 1
`N-`	: from `N`'th byte, character or field, to end of line
`N-M`	: from `N`'th to `M`'th (included) byte, character or field
`-M`	: from first to `M`'th (included) byte, character or field

5.3 Word Count

cmd	opt	args	details
`wc`	`-m`	*<file name>*	Print the character count of a file.
`wc`	`-l`	*<file name>*	Print the line count of a file.
`wc`	`-w`	*<file name>*	Print the word count of a file.

5.4 Sort

cmd	opt	args	details
`sort`		*<file name>*	Sorts and prints lines in file alphabetically.
`sort`	`-r`	*<file name>*	Sorts and prints lines in file alphabetically in reverse order.
`sort`	`-n`	*<file name>*	Sorts and prints lines in file numerically.
`sort`	`-k3`	*<file name>*	Sorts and prints lines in file based on the 3rd column (`k3`).
`sort`	`-o`	*<output file name> <input file name>*	Sorts and outputs lines from a file alphabetically into another.

5.5 Concatenate

The `cat` tool can display text, copy text from 1 or more sources to a new document or append to the end of an existing one.

cmd	opt	args	details
`cat`		*<filename ...>*	Prints out content of file(s).
`cat`		*<filename ...>* > *<output filename>*	Create new output file and copy content of source file(s) to it.
`cat`		*<filename ...>* » *<output filename>*	Copy and append content of source file(s) to an output file.
`cat`	`-n`	*<filename ...>*	Prints out content of file(s) with a line number.
`cat`	`-s`	*<filename ...>*	Prints out content of file(s) skipping empty lines.

The `tac` tool is used to concatenate and print files in **reverse**.

cmd	opt	args	details
`tac`		*<filename ...>*	Prints out content of file(s) in reverse.
`tac`	`-b`	*<filename ...>*	Attach the separator (default is a newline) before instead of after.
`tac`	`-r`	*<filename ...>*	Interpret the separator as a regular expression.
`tac`	`-s`	*<str> <filename ...>*	Use the string `str` as the separator insead of newline.

6 System variables

cmd	details
`env`	Allows for running another program in a custom environment without modifying the current one.
`printenv`	Prints *environment* variable(s).
`set`	Sets/unsets *shell* variables. Without an argument it will print a list of all variables and shell functions.
`unset`	Deletes *shell* and *environment* variables.
`export`	Sets *environment* variables.
`echo`	Prints value of a given variable* (don't forget the $ before the key - e.g.: `echo $KEY`).

* To check if the key is an *environment* variable, use `printenv`. Its output will be empty if the key is not an *environment* variable.

6.1 Syntax

Names of variables are **case-sensitive**. Note that spaces cannot be used in un-quoted (`''`, `""`) values and that multiple values assigned to a single key must be separated by colon (`:`).

Format

`KEY=value`	Value
`KEY="some value"`	String value
`KEY='some value'`	String value
`KEY=value1:value2:valueN`	Multiple values

6.2 Environment variables

Environment variables are available system-wide and are inherited by all spawned child processes and shells.

Conventionally, environment variables have their names in **uppercase**. E.g.: `MY_NAME='John Smith'`

6.2.1 Persistence

Environment variables can be made to persist between sessions, whether for the same user, multiple users on the same system or all users on a bash login shell (profile).

User Add an `export` line at the end of the user's `.bashrc` file (located in the `$HOME/` directory) and save it. To reload your `.bashrc` configuration use: `source  /.bashrc`.
E.g.: `export MY_NAME='John Smith'`

System-wide Add the variable's key-value pair in the `/etc/environment` file on a new line.
E.g.: `MY_NAME='John Smith'`

Bash profile Add an `export` line at the end of the `/etc/profile` file.
E.g.: `export MY_NAME='John Smith'`

6.3 Shell variables

Shell variables are ones that are only apply to the current shell instance. Each shell (`bash`, `zsh`, `fish`, ...) has its own set of internal variables.

7 Maths in the terminal

Doing basic arithmetic and boolean evaluations in the shell can be done with either the `expr` expression utility or the native BASH shell syntax. Note that maths can be done in `awk` as well if you want to take that route.

Values can be substituted with variable names from previous declaration whose values are of a numbered type. **Just remember to add the dollar sign** (`$`), which means in this context "*value of*", **before the name of the variable** (e.g.: `expr 3 + $my_var`).

7.1 Evaluating expressions (`expr`)

Operator	Description	`expr`
Arithmetic		
`+`	Addition	`expr 2 + 3`
`-`	Subtraction	`expr 3 - 2`
`*`	Multiplication	`expr 3 \* 2`
`/`	Division	`expr 3 / 2`
`%`	Remainder	`expr 3 % 2`
Relational		
`==`	Equality	`expr 3 = 3`
`!=`	Not Equality	`expr 3 != 4`
`>`	Larger than	`expr 5 \> 3`
`<`	Smaller than	`expr 3 \< 5`
`>=`	Larger/equal than	`expr 5 \>= 3`
`<=`	Smaller/equal than	`expr 3 \<= 5`
Other		
`match`	Match string with a **reg**ular **ex**pression	`expr match $str $regex`
`substr`	Sub-string (**pos**ition counted from 1)	`expr substr $str $pos $length`
`length`	String length	`expr length $str`
`index`	Position of first **c**haracter match or 0	`expr index $str $c`

7.2 Floating point calculations (`bc`)

As well as an interactive command line calculator, `bc` allows calculations to be piped into it. It makes this particularly useful when dealing with **floating point** calculations in either the shell or shell scripts.

To pipe just `echo` the calculations in quotations to `bc`.

Examples

```
echo '2 + 3' | bc
echo '7 % 2' | bc
echo '7 / 2' | bc
echo '(5 + 1) * 2' | bc
```

7.3 Prime factors (`factor`)

The `factor` utility can be used decompose a given integer into a list of prime factors.

7.4 Bash operators

Bash offers all basic operators as well as relational, logic, and bitwise.

For Bash, double brackets '`(( ))`' are used to:

a) enable arithmetic operations,

b) use relational and logical operators without the `test`[2] utility (e.g.: `(( 1 + 1 ))`),

c) do without the dollar sign `$` on integers and array variables (e.g.: `(( a + arr[0] ))`).

There are 2 ways to **print** a result:

1. `echo`-ing the expression (e.g.: `echo $((5 + 3))`), or
2. assign a variable to the result (e.g.: `sum=$((5 + 3))`) and then print that (e.g.: `echo $sum`).

Operator	Description	BASH
	Arithmetic	
`+`	Addition	`(( 2 + 3 ))`
`-`	Subtraction	`(( 3 - 2 ))`
`++`	Increment	`(( var++ ))`
`--`	Decrement	`(( var-- ))`
`*`	Multiplication	`(( 2 * 3 ))`
`/`	Division	`(( 3 / 2))`
`%`	Remainder	`(( 3 % 2 ))`
x^e	Exponent	`(( var**2 ))`
	Relational	
`==`	Equality	`(( 3 == 3 ))`
`!=`	Not Equal	`(( 3 != 4 ))`
`>`	Greater than	`(( 5 > 3 ))`
`<`	Lesser than	`(( 3 < 5 ))`
`>=`	Greater/equal than	`(( 5 >= 3 ))`
`<=`	Lesser/equal than	`(( 3 <= 5 ))`
	Logical	
`&&`	AND	`(( $a && $b ))`
`||`	OR	`(( $a || $b ))`
`!`	Not/Negate	`(( !$a ))`
	Bitwise	
`&`	Bitwise AND	`(( 3 & 3 ))`
`|`	Bitwise OR	`(( 3 | 4 ))`
`^`	Bitwise XOR	`(( 5 ^ 3 ))`
`~`	Bitwise complement	`(( 3 ~ 5 ))`
`<<`	Left shift	`(( 5 << 3 ))`
`>>`	Right shift	`(( 3 >> 5 ))`

[2] see section 12.5 The `test` command and its operators

8 Users and Groups

8.1 Users

cmd	opt	args	details
`useradd`		*<username>*	Adds a username.
`useradd`	`-m`		Create user directory as `/home/username`.
`useradd`	`-g`	*<group>*	Set the initial login group for a username.
`useradd`	`-G`	*<group(s)> <username>*	Add membership to supplementary group(s) (no spaces, seprated with commas) for a username.
`passwd`		*<username>*	Sets a password for username.
`usermod`	`-a -G`	*<groups> <username>*	Append user membership to group(s).
`usermod`	`-d`	*<path>* `-m` *<username>*	Change user's home directory.*
`usermod`	`-l`	*<new username> <old username>*	Changes a user's login name.*
`userdel`		*<username>*	Delete user account.
`userdel`	`-r`	*<username>*	Delete user account as well as its home directory and mail spool.

* Some care must be taken when doing these. See Arch Linux's WIKI page about it for more information.

Example

```
useradd -m -g users -G wheel,sudo -s /bin/bash $USER
```

8.2 Groups

cmd	opt	args	details
`groups`			Shows current user's group memberships.
`groups`		*<username>*	Shows user's group memberships.
`id`			Shows current user's group memberships inc. UIDs and GIDs.
`id`		*<username>*	Shows user's group memberships inc. UIDs and GIDs.
`groupadd`		*<group>*	Create a new group.
`gpasswd`	`-a`	*<username> <group>*	Add user to group.
`gpasswd`	`-d`	*<username> <group>*	Remove user from group.
`groupmod`	`-n`	*<new group> <old group>*	Rename a group (will preserve the GID).
`groupdel`		*<group>*	Delete a group.
`gpasswd`	`-d`	*<username> <group>*	Remove user membership from group.
`grpck`			Check integrity of the system's group files.

A list of the most common groups found in Linux systems is available in appendix C.

8.3 Switch User (a.k.a. Substitute User)

cmd	opt	args	details
`su`			Switch to user `root` and its default environment.
`su`		*<username>*	Switch a different user and keeps current user's environment.
`su`	`-` / `-l` / `--login`	*<username>*	Switch a different user and its default environment.

8.4 Running as root (Sudoers)

`sudo` enables execution of restricted commands (root) by users that have been granted that access. Unlike `su`, a user does not require knowing the root password.

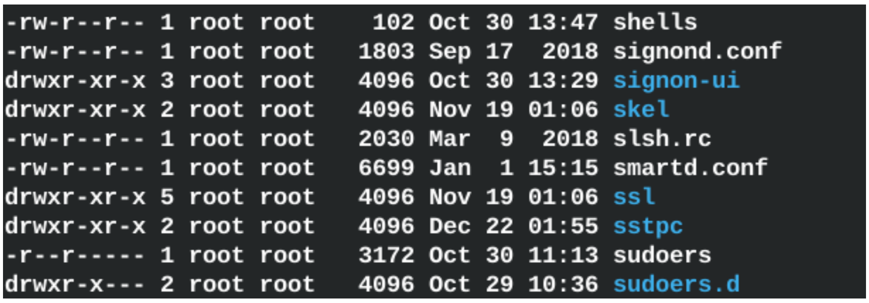
cmd	opt	args	details
`sudo`		*<command>*	Execute a command with elevated privileges.
`sudo`	`-ll`		Print current sudo configuration.
`sudo`	`-lU`	*<username>*	Print current sudo configuration for a specific user.

Run `visudo` (`/usr/sbin/visudo`) to modify the configuration. This needs to be executed from the root account or with, ironically, `sudo` (if you have elevated privileges already).

If you're feeling adventurous you could open the configuration file located in `/etc/sudoers` directly using another editor but that will now check for potential syntax errors and, thus, might break things. That is not recommended.

For information about the configuration file run `man sudoers` or check out the sudo manual pages.

8.5 File and Directory permission



```
-rw-r--r-- 1 root root    102 Oct 30 13:47 shells
-rw-r--r-- 1 root root   1803 Sep 17  2018 signond.conf
drwxr-xr-x 3 root root   4096 Oct 30 13:29 signon-ui
drwxr-xr-x 2 root root   4096 Nov 19 01:06 skel
-rw-r--r-- 1 root root   2030 Mar  9  2018 slsh.rc
-rw-r--r-- 1 root root   6699 Jan  1 15:15 smartd.conf
drwxr-xr-x 5 root root   4096 Nov 19 01:06 ssl
drwxr-xr-x 2 root root   4096 Dec 22 01:55 sstpc
-r--r----- 1 root root   3172 Oct 30 11:13 sudoers
drwxr-x--- 2 root root   4096 Oct 29 10:36 sudoers.d
```

Figure 1: Sample output from `ls -l`

All files and directories in Linux have permissions to prevent people from accessing each other's files on the machine. These permissions can be viewed with `ls -l` (see figure 1).

The columns are:

1. 10/11 character section for type and security,
2. Number of links,
3. Owner of the file,
4. Group owner of the file,
5. Size of the file in bytes,
6. Date and time of last modification,
7. File name.

8.5.1 Type and security descriptor

The type and access rights to a file is characterised by a 10/11 character long descriptor divided as such:

1	2	3	4	5	6	7	8	9	10	11
-	r	w	x	r	w	x	r	w	x	+

Character(s)	Description
1	**Type descriptor** for the entry.
2 → 5	File permissions that the **user (owner)** has.
5 → 7	File permissions that the **group** has.
8 → 10	File permissions that all the **other users** have.
11	(Optional) **Alternate access method**.

In summary; owner file permissions will only affect the owner of the file, group permissions will affect all users assigned to that group and, finally, the ‘others’ permissions affect every other users on that system.

[1] Type descriptor

-	file
d	directory
b	block file
c	character device file
p	named/unnamed pipe file
l	symbolic link file
s	socket file

[11] Alternate access methods

	None
.	Security context, no alt. access
+	Multiple access methods[a]

[a] e.g.: Access Control Lists

<table>
<tr><th>Permission</th><th>Character</th><th>For a file</th><th>For a directory</th></tr>
<tr><td>Read</td><td>-</td><td colspan="2">Content <b>cannot</b> be seen.</td></tr>
<tr><td></td><td>r</td><td colspan="2">Content <b>can</b> be seen.</td></tr>
<tr><td>Write</td><td>-</td><td colspan="2">Content <b>cannot</b> be altered in any way.</td></tr>
<tr><td></td><td>w</td><td colspan="2">Content <b>can</b> be altered.</td></tr>
<tr><td>Execute</td><td>-</td><td>The file <b>cannot</b> be executed.</td><td>The directory <b>cannot</b> be changed to.</td></tr>
<tr><td></td><td>x</td><td>The file <b>can</b> be executed.</td><td>Navigation to the directory available[3].</td></tr>
<tr><td></td><td>s</td><td colspan="2">Set the setuid[4] (for users) or setgid[4]. (for groups) bit. The x flag is set.</td></tr>
<tr><td></td><td></td><td>The file is executed with the file’s owner and/or group privileges.</td><td>When the setgid flag is set, the new files created inside the directory inherits its GID instead of the primary GID of the user who created the file. setuid has no effect.</td></tr>
<tr><td></td><td>S</td><td colspan="2">Same as s but the x flag is not set.</td></tr>
<tr><td></td><td></td><td>Rarely used.</td><td>Useless.</td></tr>
<tr><td></td><td>t</td><td colspan="2">If in the other users permissions it sets the sticky bit[5]. The x flag is also set.</td></tr>
<tr><td></td><td></td><td>Useless.</td><td>See footnote 5.</td></tr>
<tr><td></td><td>T</td><td colspan="2">Same as t but the x flag is not set.</td></tr>
<tr><td></td><td></td><td>Rarely used.</td><td>Useless.</td></tr>
</table>

[3] Using cd.

[4] Allow users to run an executable with the permissions of that executable’s owner or group.

[5] Stops non-owning users with write permissions to a folder to delete it or its content. Only the owner that created it or an administrator (e.g.: root) can delete it.

8.5.2 Changing permissions

There is 2 methods available with `chmod` for changing permissions: textual and numerical.

Text Method

The `chmod` syntax is as such: `$ chmod <who>=<permission(s)> <path>`

The 'who' argument can be a singular (e.g.: `u=`, `g=`, `o=` or `a=`) or an aggregate (e.g.: `uo=`, `ug=`, `ugo=`, etc...).

cmd	opt	args	details
`chmod`		*u=<permissions> <path>*	User
`chmod`		*g=<permissions> <path>*	Group
`chmod`		*o=<permissions> <path>*	Other users
`chmod`		*a=<permissions> <path>*	All (users and groups). Same as ' `ugo` '.

Permissions can be given in their character forms as shown in section 8.5.1 above. Like the 'who' argument, the characters can be combined (e.g.: `g=rwx`).

To copy permission over just use the letter from which to copy from as the permission. For example: `chmod g=u somefile.txt` will copy the owner/user's permissions to the group's.

Adding and removing permissions can be done with the `+` and `-` characters respectively. For example: `chmod ug+x script.sh` will add executable permissions for both owner/user and group.

Numerical Method

The `chmod` syntax is as such: `$ chmod <value> <path>`

The value must be either 3 or 4 digits long. The first 3 digits are for the permission values where the `r`/`w`/`x` values for each access type is summed up. The 4th digit is used only when a flag needs to be set (see right).

Permission values

read (`r`)	= 4
write (`w`)	= 2
execute (`x`)	= 1
none (`-`)	= 0

For example:

Owner: `rwx` = $4+2+1$ = **7**

Group: `r-x` = $4+0+1$ = **5**

Others: `r-x` = $4+0+0$ = **4**

Flag: none so **0** or omit.

owner	group	others	flag
7	5	4	0

```
$ chmod 754 script.sh
```

Flag values

`setuid`	= 4
`setgid`	= 2
sticky bit	= 1
none	= 0

cmd	opt	args	details
`stat`	`-c %a`	*<path>*	View the existing permissions of a file or directory in numeric form

Change permissions recursively

Finally, to change all content in a folder including any subfolder in the hierarchy, a recursive option is available:

cmd	opt	args	details
`chmod`	`-R`	*... <path>*	Permissions are applied recursively from path given.

8.5.3 Changing ownership

`chown` changes the owner of a file or directory.

cmd	opt	args	details
`chown`		*<new user/owner> <path>*	Change owning user of a file.
`chown`		*:<new group> <path>*	Change group of a file.
`chown`		*<new user>:<new group> <path>*	Change owning user **and** group of a file.

Note

- `chown` needs `root` privileges (or `sudo` equivalent).
- `chown` always clears the `setuid` and `setgid` bits.
- Users (except `root`) cannot use `chown` to pass ownership of files they own to other users.

Change ownerships recursively

cmd	opt	args	details
`chown`	`-R`	*... <path>*	Ownership changes are applied recursively from path given.

8.5.4 Access Control Lists (ACL)

Access Control Lists provide an additional permission framework which allows for flexible permissions to be set for any user/group to any file.

Some distributions will not have this enabled by default (like Arch). As it is a dependency for `systemd`, it should already be installed. To enable it, the filesystem must be mounted with the `acl` option.

A detailed explanation on how to do that and how to use ACL in practice is available in the Arch Linux WIKI page.

9 System and resources

9.1 Kernel

cmd	opt	args	details
`uname`	`-a`		About the current kernel: all info
`uname`	`-v`		About the current kernel: version
`uname`	`-r`		About the current kernel: release
`shutdown`	`now`		Initiate system shutdown now.
`shutdown`	`-r now`		Restart system now.

9.2 Users

cmd	opt	args	details
`w`			Show who is logged on and what they are doing.
`w`		*<user>*	Show what a particular user is doing.

9.3 Processes

cmd	opt	args	details
`ps`	`-aux`		**P**rocess **S**napshot of all processes

To see a particular process' snapshot (filter):

```
$ ps -aux | grep <process name>
```

9.4 Disks

cmd	opt	args	details
`lsblk`	`-f`		List all devices and show what filesystem are used in each.
`df`	`-ah`	*<device>*	**D**isk **F**ree: show amount of free space on device (current device if <device> is omitted.)
`du`	`-sh`	*<folder>*	**D**isk **U**sage (disk usage of a directory)
`mount`			Show all the currently **mount**ed points in the system.
`mount`		*<device> <folder>*	**Mount**s a device to a folder mount point.
`umount`		*<device>*/*<folder>*	**Unmount**s a device by its name or folder mount point.
`blkid`			Prints **bl**oc**k** device attributes/**id**s (requires elevated privileges).

9.5 Network and ports

See section 10.1 (Device and local network information).

9.6 Monitoring

Here's some basic monitoring and informational tools included in most Linux distros:

cmd	opt	args	details
`top`			Display Linux processes.
`uptime`			Shows how long the system has been up and running.
`vmstat`			Reports virtual memory statistics.
`lsof` [6]			Displays information about files open to Unix processes
`iotop` [7]			Displays information about processes' input/output to devices

10 Networking

10.1 Device and local network information

A variety of tools to get information on the local system's network state and configuration as well as modify the latter exist, for the most part, in the base installation of all Linux distros. Any missing tools can usually be installed via the package manager.

[6] Not all Linux distributions have lsof so it may need to be installed separately with the package manager.

[7] Not always installed. On Arch, install the `iotop` package.

cmd	opt	args	details
`hostid`			Prints the numeric identifier for the current host.
`hostname`			Prints or sets the name for the current host.
`ip`	`addr`		Show information for the network devices (replaces `ifconfig`[1]).
`ip`	`addr show`	*<device id>*	Show information for a particular network device (e.g.: `eth0`).
`iw`			Used to configure wireless network interface (replaces `iwconfig`).
`route`			Shows and manipulates current IP routing table.
`ss`	`-tuapn`		Check open ports what processes use them (replaces `netstat`[1]).
`arp`[1]			Allows to view or add content in the linux kernel's **A**ddress **R**esolution **P**rotocol table.

A map of network services can be found in /etc/services ("`cat /etc/sercices | less`" to browse). To see the actual status of the system's installed services use: `service --status-all`.

10.2 Remote connectivity and troubleshooting

More tools are listed here that deal with network connectivity (LAN and WAN) and can help troubleshoot problems related to this.

cmd	opt	args	details
`tracepath`		*<address>*	Prints the path take from an IP network to a given host. Less fancy equivalent to `traceroute` and does not require root privileges.
`traceroute`[2]		*<address>*	Prints the path take from an IP network to a given host.
`ping`	`-c`	*<n> <address>*	Check connectivity by sending *n* echo packets to a network destination's address.
`mtr`[2]		*<address>*	Combines `ping` and `tracepath` into a single command.
`host`[3]		*<address>*	Performs DNS lookups.
`dig`[3]		*<domain name>*	The **D**omain **I**nformation **G**roper queries DNS and helps troubleshoot related issues.
`nslookup`[3]		*<address>*	Query internet **n**ame **s**ervers. Interactive without the argument.
`whois`[2]		*<website>*	Query and prints the WHOIS data for a website.

10.3 Downloading files from the internet

There are 2 most used utilities to grab files from the internet: `curl` and `wget`.

cmd	opt	args	details
`curl`	`-O`	*<file URL>*	Download a file from the internet and save it with the same file name as the remote version.
`wget`		*<file URL>*	Download a file from the internet.

[1] Part of the `net-tools` package in Arch Linux. Note that for other distros these tools are not always installed by default.
[2] Not always installed by default.
[3] Part of the `dns-utils` package in Arch Linux. Note that for other distros these tools are not always installed by default.

10.4 Secure Shell (SSH)

SSH aims to provides a secure encrypted connection between two hosts over an insecure network. With it you can login to another networked machine, transfer files between the guest and the host and execute commands on the remote machine. There are a wide array of command line utilities centred around SSH. Here's a summary of those:

cmd	details
`ssh`	SSH Client.
`ssh-keygen`	Creates a key pair for public key authentication.
`ssh-copy-id`	Configures a public key as authorized on a server.
`ssh-agent`	Holds the private keys for single sign-on.
`ssh-add`	Add keys to the SSH agent
`scp`	RCP file transfer client.
`sftp`	FTP file transfer client.
`sshd`	OpenSSH server (daemon).

The daemon service `sshd` takes its configuration from `/etc/ssh/sshd_config` whilst the host `ssh` configuration is taken from the following in order:

1. Command line options,
2. User configuration file (`~/.ssh/config`),
3. System configuration file (`/etc/ssh/ssh_config`).

For all the nitty-gritty details check out the OpenSSH Manual.

10.4.1 Connecting to a remote SSH server

Connecting to a remote SSH server requires at the very least the target's address either in IP or domain name format. If the user account on the remote system is different than the local one, a valid username must be provided as well.

cmd	details
`ssh remote-host`	Connect to host with same currently used local username.

`ssh username@remote-host`
`ssh -l username remote-host`
	Connect to host with different username.

`ssh -p port remote-host`	Connect to host with a port number[8].
`ssh -C remote-host`	Connect to host with Compression enabled.

To **exit** from the SSH session just type '`exit`'.

[8] SSH runs on TCP/IP port 22 by default.

Troubleshooting

If there are issues connecting to the host first check that:

- the host has the server daemon (`sshd`) running,
- the client has `ssh` installed,
- the host IP address is correct,
- the host's `sshd` server daemon's listening port is not begin blocked (firewall) or forwarded incorrectly,
- the username and password used are correct.

HINT: To debug the connection use the verbose mode to get more information: `ssh -v remote-host`

10.4.2 Running remote commands/scripts

Executing commands

To execute a command on a remote system is simple:

```
ssh username@remote-host 'COMMAND'
```

There are 3 ways to execute multiple commands:

1. `ssh username@remote-host 'COMMAND1; COMMAND2; COMMAND3'`
2. `ssh username@remote-host 'COMMAND1 | COMMAND2 | COMMAND3'`

3.
   ```
   ssh username@remote-host << EOF
   COMMAND1
   COMMAND2
   COMMAND3
   EOF
   ```

Executing local shell scripts

Run a local script on the remote host:

```
ssh username@remote-host 'bash -s' < SCRIPT
```

And with arguments:

```
ssh username@remote-host 'bash -s' -- < SCRIPT --ARG
```

10.4.3 Copying files

The `scp` command is used to copy files between a local system a the remote ssh server. The syntax is as follows:

Copy local file to remote system:

```
scp <local file path> remote_host:<remote folder>
```

Copy remote file to local system:

```
scp remote_host:<remote file path> <local folder>
```

Examples

Local → remote:

```
scp ~/Documents/diary.txt root@192.168.1.25:/var/tmp/
```

Remote → local:

```
scp root@192.168.1.25:/var/log/sddm.log ~/rlogs/
```

A bit of trickery with the `tar` command (using `bzip2`) is required to copy entire folders:

Copy local folder to remote system:

```
tar -cvj <local folder> | ssh remote-host "tar -xj -C <remote folder>"
```

Copy remote folder to an archive in local system:

```
ssh user@remote-host "tar -jcf - <backup path>" > backup-name.tar.bz2
```

For backing up[9] local folders recursively to a remote "backup" server, `rsync` can be used instead:

```
rsync -az <local folder> remote-host:backup/
```

10.4.4 Starting the server daemon

To manage the execution for the `sshd` server on a `systemd` based Linux distro type in the console:

cmd	details
`systemctl status sshd`	Checks the `sshd` daemon status.
`systemctl start sshd`	Starts the `sshd` daemon.
`systemctl stop sshd`	Stops the `sshd` daemon.
`systemctl restart sshd`	Restarts the `sshd` daemon.
`systemctl enable sshd`	Enable auto-start at system boot time.
`systemctl disable sshd`	Disable auto-start at system boot time.

Be sure to go through the configuration prior starting (`/etc/ssh/ssh_config`). If you change the configuration whilst `sshd` is running you will need to restart it.

10.4.5 SSH Keys

SSH keys enable authentication between a client and a server without the need to have passwords.

1. Generate keys on the client: `ssh-keygen -t rsa`
2. Copy the public key to the remote system: `ssh-copy-id remote-host`

(1) The public key can be found as `~/.ssh/id_rsa.pub` and the private key `~/.ssh/id_rsa`.

(2) An SSH session to the remote system will be started with a username/password authentication method. Once validated the public key will be copied and future logins won't require a password.

[9] See "11 Backup" for more on the subject.

Disabling password authentication on the server

Uncomment and modify the following line in the `/etc/ssh/ssh_config` file:

`#PasswordAuthentication yes` → `PasswordAuthentication no`

Then make sure these lines are present and set to their defaults as such:

```
PubkeyAuthentication yes
ChallengeResponseAuthentication no
```

Only do the above if you have a working SSH key-based authentication with the server in place!!! Otherwise you won't be able to log-in.

10.4.6 Secure File Transfer Protocol (SFTP)

A much better alternative to the humble `ftp`[10] command, this provides a much more secure means to transfer files across a network.

The syntax to login into a remote host is simply:

cmd	details
`sftp username@remote-host`	Connect to host.
`sftp -oPort=<port> username@remote-host`	Connect to host on a specified *port* (e.g.: *22*).
`sftp username@remote-host:<folder>`	Connect to host and begin session at given folder path.

Once connected and authenticated, the `sftp` interactive prompt will appear. To get a list of the available commands just type ' `help` ' or ' `?` '. To quit the session type either ' `exit` ' or ' `bye` '.

Be aware that there is also a "Batch" mode enabling scripted interactions from a file and an "automatic retrieval" mode for just downloading files quickly.

11 Backup

Remote **sync** (`rsync` [11]) is a network capable file synchronisation tool. For copying files across a network it is preferable to `scp` [12] as it is more efficient and bandwidth friendly. When synchronising files it only copy the differences between them instead of the entirety.

There is a myriad of options, like most command line utilities, so this will just be a basic "minimum to get things running" description of `rsync` .

The basic syntax is: `rsync <option(s)> <source> <destination>`

[10] Seriously, don't use the old `ftp` command unless, at least, you are on a trusted local network with no connection to the outside (internet). It transmits authentication in plain text.

[11] Not always installed by default.

[12] See "10.4 Secure Shell (SSH)"

Note: Trailing directory slash ‘ / ’

When dealing with directories, the trailing / matters.

`<source> <destination>`
`<source> <destination>/`
	copy/sync source into destination

`<source>/ <destination>`
`<source>/ <destination>/`
	copy/sync **content** of source into destination

11.1 Commonly used options

The options in the following table don’t require special option-specific arguments and, like most commands, single character options can be combined (e.g.: `-hrv`).

opt	args	details
`-a`		Archive mode - recursively copies files and preserves their properties on copy).
`-A`		Preserves Access Control List - good for system backups.
`-b`		Create a backup.
`-c`		Skips files based on checksums instead of modification time and size.
`-e`	`<rsh>`	Specify what remote shell (`rsh`) to use (e.g.: `ssh`).
`-h`		Outputs numbers in human readable format.
`-m`		Deletes any copied/synchronised empty directories at destination (`--prune-empty-dirs`).
`-P`		Same as `--partial` and `--progress` .
`-q`		Quiet output (no info except errors).
`-r`		Recursively copy files from the directory (timestamps/permissions **not** preserved).
`-v`		Verbose output (more info).
`-X`		Preserves eXtended attributes - good for system backups.
`-z`		Compress (zips) file data during transfer.
`--partial`		Allows resume on operation that were interrupted.
`--progress`		Shows copy/synchronisation progression.
`--delete`		Deletes any superfluous destination files if they are not in source any longer.
`--include=`	*’filter’*	Include all files/directories that match the filter (see note).
`--exclude=`	*’filter’*	Exclude all files/directories that match the filter (see note).

Note: `--include` / `--exclude`

1. Filter can use the wildcard (*) character. For example: `'.*'` would match all dot files/directories. Or, `'*.odt'` would match all files with an ‘odt’ extension.
2. `--include` and `--exclude` work in tandem. Meaning that in order to only have a certain file filter **exclusively**, everything else must be excluded. e.g.: Just text documents in a directory (non-recursive) would be `--include='*.txt' --exclude='*'`
3. To combine multiple filters just format them as such: `{"filter1","filter2", ..., "filtern"}`

11.2 Local backups

Example 1: Copy/Sync a single file

Here we just need to copy/update a file so that both the source and destination are the same. None of the special file properties need to be conserved.

```
rsync some-file.tar.gz ~/Backups/
```

Example 2: Copy/Sync an entire folder

Here we want to update the archive of a folder and its content whilst preserving all file/folder properties such as groups, owner, permissions and modification times. Special files like symlinks need to be also preserved inline with a proper backup. To do that we need:

- copy any sub folders and content in the directory tree like an 'archive' (`-a`),
- remove at the destination any file/folder that doesn't exist any longer in the source (`-m`).

This results in the following syntax:

```
rsync -am ~/Documents ~/Backups/
```

Example 3: Copy/Sync all the jpg images in the " `~/Pictures/` " directory tree

Here we are looking to make a new backup of all the images of type `jpg` that can be found in the Picture folder and any sub folders recursively. The "`Pictures`" backup should also be in a dedicated directory inside of the "`Backups`" folder. For that to work we need to do the following:

- make sure to remove any empty directories created at destination as a by-product of the recursive archival process (`m`),
- include the directories (`'*/'`) so that the recursion works,
- include any files with the `jpg` file extension (`'*.jpg'`),
- exclude everything else (`--exclude='*'`),
- not have a trailing slash on the source directory (`~/Pictures`).

This results in the following syntax:

```
rsync -am --prune-empty-dirs --include={'*/','*.jpg'} --exclude='*' ~/Pictures
~/Backups/
```

Example 4: Backup entire system to an external drive

To make a backup of the entire local system (`/`) on a mounted external drive requires being mindful of a couple of things:

(a) Not all directories/files should be copied such as temporary files/folders, hardware related files, and mount points for other drives (see appendix B ”Linux directory structure” for more details).

(b) The destination directory (mount point for the backup drive) **must be excluded** lest `rsync` run into an infinite loop. For this example, let’s assume we’ve mounted the backup drive in `/mnt/backup` .

(c) As it is the entire system we will need to either be running form the `root` account or have adequate elevated `sudo` privileges to do so.

Here are the properties that need to be covered by the command:

1. make sure to conserve all file properties, attributes and access control lists (`-a` , `-A` , `-X`),
2. show information (`-v`) and progress (`--progress`) as it will be a large process to backup everything,
3. exclude all special file/directories from the backup (`/dev/*` , `/proc/*` , `/sys/*` , `/tmp/*` , `/run/*` , `/mnt/*` , `/media/*` , `/lost+found` [a])

All of this leaves us with the following syntax:

```
rsync -aAXv --progress --exclude={"/dev/*","/proc/*","/sys/*","/tmp/*","/run/*",
"/mnt/*","/media/*","/lost+found"} / /mnt/backup
```

[a] The `lost+found` directories are special `fsck` folders for recovered lost file (orphaned inodes).

11.3 Remote backups

Remote backups are similar in syntax as to the local ones. To connect to the remote system, `rsync` can either use it’s internal protocol or just tunnels through SSH. Credentials to the remote system must have been setup prior. The syntax for remote operations is:

local → remote: `rsync <option(s)> <local directory> username@remote-host:<remote directory>`

remote → local: `rsync <option(s)> username@remote-host:<remote directory> <local directory>`

Example 1: Copy/Sync `~/Documents` folder to a remote LAN machine

Let’s say that we want to copy/sync our `Document` folder to another machine on the local network (`192.168.1.56`) so that when we login with the same username on that (`johndoe`), we have the same documents there as in our current system. For that to work we need to do the following:

- copy the files/folders recursively with their properties (`-a` , `-X`),
- use compression to transfer the data (`-z`),
- show progress and enable resume if the network connection breaks/times-out during transfer (`-P`),
- Remove any superfluous files on the remote if they are not the local folder any longer (`--delete`).

We end up with the following syntax:

```
rsync -aXzP --delete ~/Documents/ johndoe@192.168.1.56:/home/johndoe/Documents/
```

Example 2: Copy/Sync LAN remote **`pacman`**'s cached packages to local

Say the cache on a remote machine (`192.168.1.3`) has the latest packages for an Arch based Linux distro (`/var/cache/pacman/pkg/`) and, to avoid re-downloading all of these from the internet for a local system's update, we just copy them over. For that to work we need to do the following:

- copy the files/folders recursively with their properties (`-a`, `-X`),
- use compression to transfer the data (`-z`),
- show progress and enable resume if the network connection breaks/times-out during transfer (`-P`),
- copy as `root` since these are system packages.

This leaves us with:

```
rsync -aXzP root@192.168.1.3:/var/cache/pacman/pkg/ /var/cache/pacman/pkg/
```

Example 3: Backup a local file **`jdbp.tar.gz`** to WAN remote backup system

In this scenario we are creating a up-to-date copy of the `Pictures` directory on a remote server on the internet (`backups.jdoe.dev`). For this we need to do the following:

- copy the file with its extended properties intact (`-X`),
- use compression to transfer the data (`-z`),
- show progress and enable resume if the network connection breaks/times-out during transfer (`-P`),
- use `ssh` on the none-default port `2514` to login/access the remote machine[a] (`-e ssh`).

The syntax becomes:

```
rsync -XzP -e 'ssh -p 2514' /home/johndoe/jdbp.tar.gz johndoe@backups.jdoe.dev:/dump/
```

[a] Assuming `ssh` credentials have already been created prior.

12 Shell Scripting

Shell scripting allows automation on any groups of commands you may want to execute with the added bonus of some common programming flow control idioms and variable storage.

This chapter introduces the basics required to get reasonably functional scripts off the ground. For an advance look at all that shell scripting can offer check out The Advanced Bash Scripting Guide by Mendel Cooper.

12.1 The script file

A shell script file is a simple text file that can be executed. To make one just create a blank text file with the `.sh` extension and make it executable (see 8.5.2 Changing permissions).

At the very top of the file remember to add this line: `#!/bin/bash`

It is a special comment (#) that shells look for and tells that a) this is a shell script and b) the kind of shell script it is. In our above case, the line informs the current shell that it should be run with Bash.

12.2 Bash special parameters

Bash provides a number of useful ”special parameters”.

Parameter	Description
`$#`	Number of positional parameters (like `argc` in C/C++’s `main()`).
`$0`	Name of the shell or shell script (like `argv[0]` in C/C++’s `main()`).
`$1`, `$2`, ... `$`*n*	Positional parameters (like `argv[1]`, ..., `argv[`*n*`]` in C/C++’s `main()`).
`$@`	Array-like construct of all positional parameters.
`$IFS`	**I**nput **f**ield **s**eparator.
`$*`	is the IFS expansion of all positional parameters.
`$-`	Current options set for the shell.
`$$`	Process ID (`pid`) of the current shell (not subshell)
`$_`	Most recent parameter
	(or the abs path of the command to start the current shell immediately after startup).
`$?`	Most recent foreground pipeline exit status (see 12.3 Exit codes).
`$!`	Process ID (`pid`) of the most recent background command.

12.3 Exit codes

All Bash built-ins, if successful, return a ’`0`’ exit code. When unsuccessful they return a non-zero status. Below is a list[13] of reserved error exit codes for Bash.

Code	Description
1	Catch-all for general errors.
2	Incorrect shell built-in usage (i.e.: invalid options, missing arguments, etc...)
126	Failed to execute invoked command.
127	Command not found.
128	Invalid argument to exit[14].
128+*n*	Fatal error ’*n*’.
130	Script terminated via Ctrl + C

When building shell scripts returning exit codes when failure occurs should be considered. A full list of the most common exit codes found in Linux is available in appendix D.

[13] Taken from Advanced Bash-Scripting Guide - Appendix E.

[14] Exit code can only be integers in the range of 0-255

12.4 Conditional constructs

((...)) The brackets evaluate the expression within. If the value of the expression is $!0$, the status returned is 0. Otherwise the status returned is 1. Also used in arithmetic.

[...] The single bracket expression is POSIX[15] so offers the best compatibility between different shells. The "`[`" is an actual command and its counterpart, "`]`", is an argument signalling the end of the expression.

[[...]] The double bracket expression, unlike the single version, is a Bash extension and thus subject to support issues based on what shell is used to run the script. It's evaluation of expressions for certain operators and word splitting rules are also a little different (`<`, `&&`, `||`, `()`, `=` and `~=`).

For more details check out Bash Reference Manual: Conditional Constructs.

12.5 The `test` command and its operators

The `test` command is used to evaluate conditional expressions, meaning that the results can either be *true* or *false*. Its syntax is as follows:

```
test <expression>
```

or

```
[ <expression> ]
```

Several built-in operators can be used with the test command. These operators can be classified into 3 groups: numeric/logical operators, string operators, file operators.

12.5.1 Numeric operators

Purpose	Operator	Description
Equality	`<a> -eq <b>`	True if integer `a` is equal to `b`.
Greater/equal than	`<a> -ge <b>`	True if integer `a` is greater than or equal to `b`.
Greater than	`<a> -gt <b>`	True if integer `a` is greater than `b`.
Lesser/equal than	`<a> -le <b>`	True if integer `a` is less than or equal to `b`.
Lesser than	`<a> -lt <b>`	True if integer `a` is less than `b`.
Not Equal	`<a> -ne <b>`	True if integer `a` is not equal to `b`.

Example: `test $a -eq $b` or `[ $a -eq $b ]`

[15] Portable Operating System Interface: set of IEEE standards for maintaining compatibility between operating systems.

12.5.2 String operators

Purpose	Operator	Description
Same	`<str a> = <str b>`	True if string a is identical to b.
Not same	`<str a> != <str b>`	True if string a is **not** identical to b.
Not null	`<str>`	True if `str` is not null.
Length > 0	`-n <str>`	True if length of `str` is greater than zero.
Length is 0	`-z <str>`	True if length of `str` is equal to zero.

Example: `test $a = $b` or `[ $a = $b ]`

12.5.3 File comparison operators

Purpose	Operator	Description
Same	`<file a> -ef <file b>`	True if a and b refer to the same device and iNode number.
Newer	`<file a> -nt <file b>`	True if a is newer (based on modification date) than b, or if a exists and b does not.
Older	`<file a> -ot <file b>`	True if a is older than b, or if b exists and a does not.

12.5.4 File state operators

Purpose	Operator	Description
Existence	`-a <file name>`	True if file exists.
Existence (Block file)	`-b <file name>`	True if **b**lock file exists (e.g.: hard drive or partition).
Existence (Character file)	`-c <file name>`	True if **c**haracter file exists (e.g.: TTY device).
Existence (Directory)	`-d <file name>`	True if **d**irectory exists.
Existence	`-e <file name>`	True if file exists (same as `-a`).
Existence (Regular file)	`-f <file name>`	True if **f**ile exists and is of regular type.
Existence (+ with `setgid`)	`-g <file name>`	True if file exists and has the `setgid` flag set.
Existence (+ owned by GID)	`-G <file name>`	True if file exists and is owned by effective group ID.
Existence (symbolic link)	`-h <file name>`	True if file exists and is a symbolic link.
Existence (+ with 'sticky' bit)	`-k <file name>`	True if file exists and has a "sticky" bit set.
Existence (symbolic link)	`-L <file name>`	True if file exists and is a symbolic **l**ink.
Existence (+ modified)	`-N <file name>`	True if file exists and modified since last read.
Existence (+ owned by UID)	`-O <file name>`	True if file exists and is **o**wned by the effective user ID.
Existence (pipe)	`-p <file name>`	True if file exists and is a named **p**ipe (FIFO).
Existence (+ readable)	`-r <file name>`	True if file exists and is **r**eadable.
Existence (+ size > 0)	`-s <file name>`	True if file exists and has a **s**ize greater than zero.
Existence (socket)	`-S <file name>`	True if file exists and is a **s**ocket.
Opened file descriptor	`-t <fd>`	True if file descriptor is open and refers to a **t**erminal.
Existence (+ with `setuserid`)	`-u <file name>`	True if file exists and `setuserid` flag is set.
Existence (+ writeable)	`-w <file name>`	True if file exists and is **w**riteable.
Existence (+ executable)	`-x <file name>`	True if file exists and is e**x**ecutable.

12.6 Variables

To assign a value to a variable simply put the variable and the value together separated with an equal sign without any spaces between (`=`). There is no need to do type declaration so for example:

```
myint=10
mystr="hello, world!"
```

To access your variable's stored value just prefix the variable's name with a dollar[16] (`$`) sign.

```
for i in {0..10}; do
    echo "$mystr"
done
```

12.7 Quotation marks and Escape character

The double quotation marks (`""`) and single quotation marks (`''`) are used to hide special characters from the shell. The doubles only hide white-spaces leaving the other special characters to be interpreted by the shell whilst the singles hide both - essentially making whatever is in between verbatim text.

The backslash (`\`) enables per-character granularity in hiding special characters. In short, it is the same as the single quotes but for single characters only.

Type	White-space	Special chars	Use(s)
`""`	Hidden	Interpreted	Assigning strings that contain 2+ words.
`''`	Hidden	Hidden	Passing command to other programs.
`\`	Hidden	Hidden	Hides a single special character from shell interpretation.

For example:

Demo script:

```
str="Hello, world!"
echo 'Testing single quotes: $str'
echo "Testing double quotes: $str"
echo "Testing backslash    :" Hello,\ world!
```

Output:

```
Testing single quotes: $str
Testing double quotes: Hello, world!
Testing backslash    : Hello, world!
```

12.7.1 Back quotes

Back quotes (``` `` ```) are used for storing or using the result of a given command.

Example:

```
dir_content=`ls`
echo "Directory content:"
echo "$dir_content"
```

[16] `$` means, in this context, '*value of*'.

12.8 Printing

There are 2 options to print content to the console: `echo` and `printf`. Both are pretty universal[17]. The `printf` functionality/utility provides more control over the output format compared to `echo`. To note that, by default, `echo` adds a newline automatically unless instructed otherwise whereas `printf` does not.

12.8.1 echo

cmd	opt	args	details
`echo`		*<string>*	Prints string.
`echo`	`-n`	*<string>*	Prints string but omit **n**ewline from the output.
`echo`	`-e`	*<string>*	Prints string and enable the function of the backslash (`\`) character.
`echo`	`-E`	*<string>*	Prints string and disable the function of the backslash (`\`) character.

12.8.2 printf

`printf` takes a first string with formatting markers and then the arguments to place into said string.

cmd	opt	args	details
`printf`		*<format> <argument(s)>*	Prints formatted string with argument(s).

The *<format>* string can contain:

1. Normal text/characters that will be printed verbatim.
2. Interpreted text/characters that are escaped with a backslash (`\`).

Sequence	Description
`\"`	Double quote.
`\NNN`	Character with octal value *NNN* 1 to 3 digit long.
`\\`	Backslash.
`\a`	Alert (BEL).
`\b`	Backspace.
`\c`	Produce no further output.
`\f`	Form feed.
`\n`	New line.
`\r`	Carriage return.
`\t`	Horizontal tab.
`\v`	Vertical tab.
`\xHH`	Hexadecimal byte value 'HH' (1 to 2 digits).
`\uHHHH`	Unicode (ISO/IEC 10646) character with hexadecimal value 'HHHH' (4 digits).
`\`	Unicode character with hexadecimal value 'HHHHHHHH' (8 digits).
`\%\%`	A single `%`.
`\b`	*<argument>* as a string with backslash escapes interpreted (except octal: `\0` or `\0NNN`)

[17] Both are built-in commands in Bash and most distros will also have a stand-alone binary of those (try '`type -a echo`' and '`type -a printf`' to verify that.)

3. Insertion specifications that describe how the *<argument(s)>* will be printed. The format of this is `%MS` where `M` is the optional **modifier** and `S` is the **specification** character (e.g.: `%4.1d`).

   A **modifier** can be composed of any of the following in order:

`-`	Left-adjust the argument conversion.
`number`	Minimum field width which can be padded when necessary (`int`).
`.`	Separator for field width and precision.
`number`	Precision that specifies the (a) max number of characters from a string, (b) digits after the decimal point of a float value or (c) minimum number of digits for an integer to be printed.
`h` or `l`	Differentiate between a short and long integers.

   The argument **conversion specification** can be any 1 of the following:

Spec. char	Description
`d`, `i`	Integer given as a decimal number.
`o`	Integer given as an unsigned octal number.
`x`, `X`	Integer given as an unsigned hexadecimal number.
`u`	Integer given as an unsigned decimal number.
`c`	Integer given as an ASCII character whose code will be used.
`s`	String.
`f`	Floating-point number (default precision of 6).
`e`, `E`	Floating-point number given in scientific notation (default precision of 6).
`p`	Memory address pointer.
`%`	Literal percent sign ("%").

Example 1

```
printf "My name is \"%s\".\nI'm %u years old." "Bart Simpson" 10
```

Composed of:

- Normal text,
- Escaped characters: literal double quotation marks (\") and new line (\n),
- A "string" insertion: %s,
- An "unsigned decimal number" insertion: %u.

Result:

```
My name is Bart Simpson.
I'm 10 years old.
```

Example 2

```
printf "Num: \'%8.2f\'\nString: \"%8.5s\"" 1234.5678 "hello world"
```

Composed of:

- Normal text,
- Escaped characters: literal double and single quotation marks (\”, \’) and new line (\n),
- An ”floating point number” insertion with a min field width of 8 and with 2 digits after the decimal point: %8.2f.
- A ”string” insertion with a min field width of 8 characters and 5 characters to be shown from the string: %8.5s,

Result:

```
Num: ' 1234.57'
String: "   hello"
```

12.9 User input

To take in input in the terminal from a user the `read` utility can be used.

cmd	opt	args	details
`read`			Reads a single line from the standard input and store it in ‘`REPLY`’.
`read`	`-a`	*ARRAY*	Takes the words read and stores them in an array ‘`ARRAY`’.
`read`	`-d`	*DELIM*	Continues reading until the first character of `DELIM` is read, rather than newline.
`read`	`-e`		Uses ‘Readline’ to obtain the line to be read.
`read`	`-i`	*TEXT*	Uses `TEXT` as the initial text for ‘Readline’.
`read`	`-n`	*NCHARS*	Returns after reading `NCHARS` characters instead of waiting for a newline (delimiter `-d` takes priority if found before `NCHARS`).
`read`	`-N`	*NCHARS*	Returns only after reading exactly `NCHARS` characters, unless an EOF is encountered or read times out (delimiters `-d` are ignored).
`read`	`-p`	*PROMPT*	Outputs the string `PROMPT` and reads the input on the same line.
`read`	`-r`		Diss-allows backslashes to escape any characters.
`read`	`-s`		Does not echo the terminal input to screen (i.e. hides what is typed).
`read`	`-t`	*TIMEOUT*	Times out and return failure if a complete line of input is not read within `TIMEOUT` seconds (if timeout is exceeded the error code will be >128).
`read`	`-u`	*FD*	Reads from a file descriptor (`FD`) instead of the standard input.

Example 1: Simple read with prompt

The script just asks for a `name` then prints a reply using the `name` given.

```
#!/bin/bash
read -p "What's your name?: " name
echo "Hello $name!"
```

Example 2: Username and password

The script just asks for a `username` then a `password`. The `password` is hidden as it is typed. A check (line 5) is made and the result is printed.

```
1 #!/bin/bash
2 read -p "Username: " username
3 read -sp "Password: " password
4
5  if [ "$username" = "root" ] && [ "$password" = "123456" ]; then
6     echo "Login is correct"
7 else
8     echo "Login is incorrect"
9  fi
```

Note: this is not a secure way of checking credential!

12.10 Flow control

12.10.1 `if`

structure

```
if [ expression1 ]; then
    #...\
elif [ expression2 ]; then
    #...
else
    #...
fi
```

example

```
if [ $a -gt $b ]; then
    echo $a " greater than " $b
elif [ $a -lt $b ]; then
    echo $a " lesser than " $b
else
    echo $a " equal to " $b
fi
```

12.10.2 `switch`

structure

```
case string in
str1)
    #...
    ;;
str2)
    #...
    ;;
*)
    #...
    ;;
esac
```

example

```
case $str in
'john')
    echo "Hello John"
    ;;
'dave')
    echo "Hello Dave!"
    ;;
*)
    echo "Hello."
    ;;
esac
```

12.10.3 for

structure

```
for var in list; do
    #...
done
```

example

```
for i in ${NAMES}; do
    echo ”Hello ${i}!”
done
```

Alternatively there is a 3 expression variation available (similar to C++/Java and the likes):

structure

```
for (( EXP1; EXP2; EXP3 )); do
    #...
done
```

example

```
for (( i=0; i<=10; i++ )); do
    echo ”Hello, world!”
done
```

12.10.4 while

structure

```
while expression; do
    #...
done

```

example

```
while [ $i −lt 10 ]; do
    echo ”i = : $i”
    i=$(( $i + 1 ))
done
```

12.10.5 until

```
until expression; do
    #...
done
```

12.10.6 shift (for positional parameter)

The shift command is used to move all values stored in the positional parameters ($1, $2, ... $*n*) to the left. The value at position $0 remains unaffected.

For example, with the following values in store:

$1 = -a
$2 = *doc1.txt*
$3 = *doc2.txt*

Shifting the values will pop the first value at $1 and move the rest 1 position left thus leaving us with:

$1 = *doc1.txt*
$2 = *doc2.txt*

It is possible to specify by how much the shift should move the values by. Just add the parameter after the command (i.e.: shift *n*).

Iterating through the parameters

```
while [ ”$1” ]; do
    #...
    shift
done
```

12.11 Functions

Functions can have any number of parameters passed to them and, within, will see those as positional parameters (`$1`, `$2`, ... `$`*n*). It works just like the ones the shell script gets from the command line but locally to the function.

structure

Declaration

```
functionName () {
    #...
}
```

Invocation

```
functionName [param1 param2 param3 ...]
```

example

Declaration

```
print () {
    echo "$1"
}
```

Invocation

```
print $str
```

12.12 Debugging and Linting

To debug/lint your script the ShellCheck tool is available as a online version as well as local (available in most major Distros repositories).

Otherwise there is the Bash debugger with a `gdb`-like command syntax.

13 Automating tasks

The ”Cron” tool enables tasks to be run on a schedule. Each user on a system has his/her own ”Cron” pool meaning that if a user sets up a scheduled task the other users will not have it run in their profile.

The ”Cron” background daemon checks the `/etc/crontab` file as well as the directories `/var/spool/cron/` and `/etc/cron.*/`. It is **not** advisable to edit these directly/manually.

cmd	opt	details
`crontab`	`-e`	Edit cron jobs for current user (see note below).
`crontab`	`-l`	List all cron jobs for current user.
`crontab`	`-r`	Remove all cron jobs for current user.

To specify another user, the `-u <username>` option can be used.

Note: `crontab` editor

Editing cron jobs uses whatever editor is specified in the environment variables `VISUAL` or `EDITOR`. If both of these are not set and the default (`vi`) is not installed there will be an error.

To set the environment variable check out ”6.2 Environment variables”.

13.1 Editing tasks

Once inside the editor, tasks can be added/removed/modified at will. The syntax is very simple:

```
m h d M w <username> /path/to/command <args>
```

Arguments (`<args>`) are optional and the username (`<username>`) is not required for the current user.

Example 1: Run `backup.sh` every day at midnight

```
0 0 * * * backup.sh
```

Example 2: Run `cleanup.sh` mon, wed, and fridays at 11:30pm

Either `30 23 * * 1,3,5 cleanup.sh`

or `30 23 * * mon,wed,fri cleanup.sh`

Example 3: Run `update.sh` every 6 hours on weekdays

Either `* */6 * * 1-5 update.sh`

or `* */6 * * mon-fri update.sh`

Example 4: Reboot system every 6 hours

```
* */6 * * * /usr/bin/reboot
```

Scheduling variables

`m`	Minute (`0` → `59`)
`h`	Hour (`0` → `23`)
`d`	Day (`0` → `31`)
`M`	Month (`0` → `12`)
`w`	Weekday (`0` → `7`)

Scheduling symbols

`*`	All possible values for field
`,`	List separator
`-`	Range separator
`/`	Step separator

Syntax Shortcuts

`@hourly`	→	`0 * * * *`
`@midnight`	→	`0 0 * * *`
`@daily`	→	`0 0 * * *`
`@weekly`	→	`0 0 * * 0`
`@monthly`	→	`0 0 1 * *`
`@annually`	→	`0 0 1 1 *`
`@yearly`	→	`0 0 1 1 *`
`@reboot`		Every startup

13.2 Allow/Deny users to schedule tasks

It is possible to restrict the use of ”Cron” for users on a system with the `/etc/cron.deny` and `/etc/cron.allow` files that act, receptively, like a blacklist and a whitelist of users.

Username can be added to these files to either deny or allow the use of the `crontab` command. By default only `cron.deny` exits. If `cron.allow` is created then **only** the users listed in it can access the `crontab` command. If both files are missing then only `root` has access.

To summarize the command access based on what file exists:

`cron.deny`	`cron.allow`	Access
×	×	Only `root` account.
✓	×	All users except those in `cron.deny` .
×	✓	Only users in `cron.allow` .
✓	✓	Only users in `cron.allow` .

14 Common scenarios

14.1 Formatting a USB stick

The table show the **native** compatibility of different filesystems. Most can be added with 3rd party packages to work on other system though with a bit of research and work.

Filesystem	Description	Linux	Mac OSX	Windows
`ext4`	Linux native format	✓	×	×
`FAT32`	Old DOS/Windows format	✓[18]	×	✓
`exFAT`	New-ish Windows format for external devices	✓[19]	✓	✓

First find out what device partition name is used for the stick (`lsblk` can show that info). For example: `sdf1`.

1. Unmount the device: `umount /dev/<device>`
2. Format device:

   ext4: `sudo mkfs.ext4 /dev/<device>`
   FAT32: `mkdosfs -F 32 -I /dev/<device>`
   exFAT: `sudo mkfs.exfat /dev/<device>`
3. Create a label:

   ext4: `sudo e2label /dev/<device> "<label>"`
   FAT32: `fatlabel /dev/<device> <label>` (uppercase, no spaces and 11 characters max)
   exFAT: `exfatlabel /dev/<device> '<label>'` (15 characters max)
4. Make permissions universal:

   ext4: `sudo chmod 777 <path to mounted drive>`
   FAT32: N/A
   exFAT: N/A

14.2 What is blocking umount?

This utility (`lsof`) is not always included in a Linux distribution so you may have to install it first.

```
$ lsof | grep <path to mounted device>
```

14.3 Remove a list of files

To remove a list of files, like for example the output of a `find` query, it needs to be piped via a `xargs` command:

```
$ find . -type f -name *.old -print0 | xargs -0 rm
```

[18] Needs the `mtools` package to be installed on Arch.

[19] Needs the `exfat-utils` package to be installed on Arch.

14.4 Piping lines from a file to a script

Example script `script.sh`:

```bash
#!/bin/bash
set -e              #break on error

if [ -p /dev/stdin ]; then
    while IFS= read line; do
        printf "${line}\n”
    done
fi
```

```
cat  source.txt | ./script.sh
```

15 Other interesting applications

These will require installing but are listed there as they can be extremely useful for specific scenarios.

Package	Description
imagemagick	A complete swiss-army knife collection of CLI based image viewing/manipulation programs (`Magick++-config` , `MagickCore-config` , `MagickWand-config` , `animate` , `compare` , `composite` , `conjure` , `convert` , `display` , `identify` , `import` , `magick` , `magick-script` , `mogrify` , `montage` , `stream`).
f3	Utilities to detect and repair counterfeit flash storage, i.e. thumb drives and memory cards with less flash than advertised. (`f3brew` , `f3fix` , `f3probe` , `f3read` , `f3write`)

16 Change log

Date	Section	Topic(s)	Change
11/02/20	Everything	original publication	-
17/02/20	5.2 Slicing and extracting	`head` and `tail`	add
17/02/20	5.5 Concatenate	`tac`	add
26/02/20	9.6 Monitoring	`iotop`	add
26/02/20	10.1 Device and local network information	How to get list of services and their status.	add
26/02/20	14.1 Formatting a USB stick	FAT32 and exFAT, compatibility table	add
10/03/20	4.1 Files and Directories	`pwd`	add
10/03/20	5.2 Slicing and extracting	`cut`	add
10/03/20	9.4 Disks	`blkid`	add
10/03/20	15 Other interesting applications	imagemagick, f3	add
24/12/21	14.4 Piping lines from a file to a script	Piping lines to a script	add
24/12/21	14.3 Remove a list of files	Find and remove resulting files	add

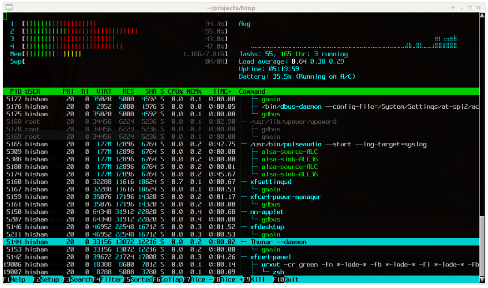
Appendices

A More monitoring tools

There are more tools available that can be installed and go beyond the basics for monitoring. They can be especially useful for system administrators and such. Here's a curated selection:

htop

htop essentially supercharges and beautifies the native `top` application. Its' available in most repositories so can be installed via your distro's package manager.



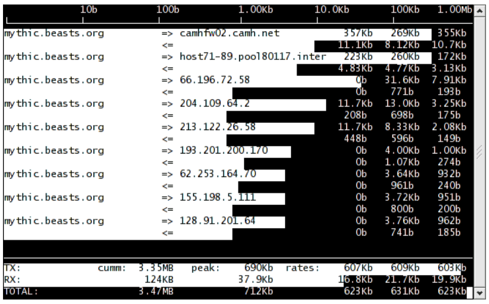
```
~/projects/htop

  1  [|||||||||||||||||||||||                         34.3%]   Avg
  2  [||||||||||||||||||||||||||||||||||||            55.0%]
  3  [|||||||||||||||||||||||||||                     43.0%]
  4  [|||||||||||||||||||||||||||||                   47.0%]
  Mem[||||||||||||||||                          1.16G/7.81G]   Tasks: 55, 165 thr; 3 running
  Swp[                                                0K/0K]   Load average: 0.64 0.38 0.29
                                                               Uptime: 05:19:59
                                                               Battery: 35.5% (Running on A/C)

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 5177 hisham     20   0 35020  5000  4592 S  0.0  0.1  0:00.00 │  ├─ gmain
 5176 hisham     20   0  2952  2080  1976 S  0.0  0.0  0:00.05 │  ├─ /bin/dbus-daemon --config-file=/System/Settings/at-spi2/ac
 5175 hisham     20   0 35020  5000  4592 S  0.0  0.1  0:00.00 │  └─ gdbus
 5168 root       20   0 34456  6224  5236 S  0.0  0.1  0:02.90 ├─ /usr/lib/upower/upowerd
 5170 root       20   0 34456  6224  5236 S  0.0  0.1  0:00.00 │  ├─ gdbus
 5169 root       20   0 34456  6224  5236 S  0.0  0.1  0:00.00 │  └─ gmain
 5165 hisham     20   0  177M 12896  6764 S  0.0  0.2  0:47.75 ├─ /usr/bin/pulseaudio --start --log-target=syslog
 5309 hisham     20   0  177M 12896  6764 S  0.0  0.2  0:00.00 │  ├─ alsa-source-ALC
 5308 hisham     20   0  177M 12896  6764 S  0.0  0.2  0:00.00 │  ├─ alsa-sink-ALC36
 5180 hisham     20   0  177M 12896  6764 S  0.0  0.2  0:00.01 │  ├─ alsa-source-ALC
 5174 hisham     20   0  177M 12896  6764 S  0.0  0.2  0:45.67 │  └─ alsa-sink-ALC36
 5160 hisham     20   0 32288 11616 10624 S  0.7  0.1  0:00.67 ├─ xfsettingsd
 5167 hisham     20   0 32288 11616 10624 S  0.0  0.1  0:00.53 │  └─ gmain
 5159 hisham     20   0 35076 17196 14320 S  0.0  0.2  0:01.17 ├─ xfce4-power-manager
 5161 hisham     20   0 35076 17196 14320 S  0.0  0.2  0:00.00 │  └─ gdbus
 5150 hisham     20   0 64348 31912 22820 S  0.0  0.4  0:00.68 ├─ nm-applet
 5207 hisham     20   0 64348 31912 22820 S  0.0  0.4  0:00.00 │  └─ gdbus
 5146 hisham     20   0 46952 22548 16712 S  0.0  0.3  0:01.52 ├─ xfdesktop
 5211 hisham     20   0 46952 22548 16712 S  0.0  0.3  0:00.53 │  └─ gmain
 5144 hisham     20   0 33156 13072 12216 S  0.0  0.2  0:00.02 ├─ Thunar --daemon
 5153 hisham     20   0 33156 13072 12216 S  0.0  0.2  0:00.00 │  └─ gmain
 5142 hisham     20   0 39672 21724 17008 S  0.0  0.3  0:04.26 ├─ xfce4-panel
19006 hisham     20   0 18388  8600  7012 S  0.0  0.1  0:00.14 │  ├─ urxvt -cr green -fn *-lode-* -fb *-lode-* -fi *-lode-* -fb
19007 hisham     20   0  8788  5088  3780 S  0.0  0.1  0:00.09 │  │  └─ zsh
F1Help  F2Setup F3SearchF4FilterF5SortedF6CollapF7Nice -F8Nice +F9Kill  F10Quit
```

iftop

"iftop does for network usage what top(1) does for CPU usage. It listens to network traffic on a named interface and displays a table of current bandwidth usage by pairs of hosts."

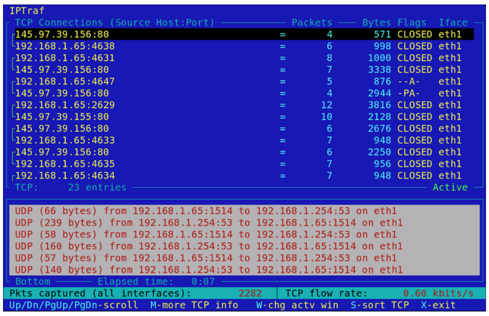


```
               10b          100b         1.00Kb       10.0Kb       100Kb    1.00Mb
└──────────────┴────────────┴────────────┴────────────┴────────────┴──────────
mythic.beasts.org          => camhfw02.camh.net          357Kb   269Kb   355Kb
                           <=                            11.1Kb  8.12Kb  10.7Kb
mythic.beasts.org          => host71-89.pool80117.inter  223Kb   260Kb   172Kb
                           <=                            4.83Kb  4.77Kb  3.13Kb
mythic.beasts.org          => 66.196.72.58                  0b   31.6Kb  7.91Kb
                           <=                               0b    771b    193b
mythic.beasts.org          => 204.109.64.2               11.7Kb  13.0Kb  3.25Kb
                           <=                             208b    698b    175b
mythic.beasts.org          => 213.122.26.58              11.7Kb  8.33Kb  2.08Kb
                           <=                             448b    596b    149b
mythic.beasts.org          => 193.201.200.170               0b   4.00Kb  1.00Kb
                           <=                               0b   1.07Kb   274b
mythic.beasts.org          => 62.253.164.70                 0b   3.64Kb   932b
                           <=                               0b    961b    240b
mythic.beasts.org          => 155.198.5.111                 0b   3.72Kb   951b
                           <=                               0b    800b    200b
mythic.beasts.org          => 128.91.201.64                 0b   3.76Kb   962b
                           <=                               0b    741b    185b

TX:             cumm:  3.35MB   peak:   690Kb  rates:   607Kb   609Kb   603Kb
RX:                     124KB          37.9Kb          16.8Kb  21.7Kb  19.9Kb
TOTAL:                 3.47MB           712Kb           623Kb   631Kb   623Kb
```

iptraf

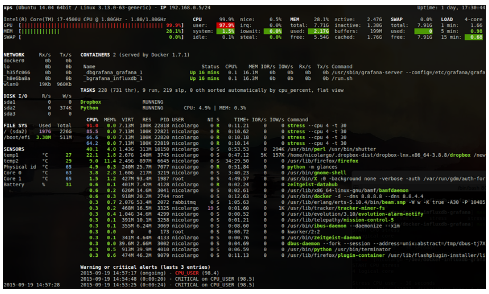
"iptraf is a console-based network statistics utility for Linux. It gathers a variety of figures such as TCP connection packet and byte counts, interface statistics and activity indicators, TCP/UDP traffic breakdowns, and LAN station packet and byte counts."



```
 IPTraf
┌ TCP Connections (Source Host:Port) ─────────── Packets ─── Bytes Flags  Iface ┐
│┌145.97.39.156:80                         =        4       571 CLOSED eth1   │
│└192.168.1.65:4638                        =        6       998 CLOSED eth1   │
│┌192.168.1.65:4631                        =        8      1000 CLOSED eth1   │
│└145.97.39.156:80                         =        7      3338 CLOSED eth1   │
│┌192.168.1.65:4647                        =        5       876 --A-   eth1   │
│└145.97.39.156:80                         =        4      2944 -PA-   eth1   │
│┌192.168.1.65:2629                        =       12      3816 CLOSED eth1   │
│└145.97.39.155:80                         =       10      2128 CLOSED eth1   │
│┌145.97.39.156:80                         =        6      2676 CLOSED eth1   │
│└192.168.1.65:4633                        =        7       948 CLOSED eth1   │
│┌145.97.39.156:80                         =        6      2250 CLOSED eth1   │
│└192.168.1.65:4635                        =        7       956 CLOSED eth1   │
│┌192.168.1.65:4634                        =        7       948 CLOSED eth1   │
└ TCP:      23 entries ──────────────────────────────────────────────── Active ┘
┌──────────────────────────────────────────────────────────────────────────────┐
│ UDP (66 bytes) from 192.168.1.65:1514 to 192.168.1.254:53 on eth1            │
│ UDP (239 bytes) from 192.168.1.254:53 to 192.168.1.65:1514 on eth1           │
│ UDP (58 bytes) from 192.168.1.65:1514 to 192.168.1.254:53 on eth1            │
│ UDP (160 bytes) from 192.168.1.254:53 to 192.168.1.65:1514 on eth1           │
│ UDP (57 bytes) from 192.168.1.65:1514 to 192.168.1.254:53 on eth1            │
│ UDP (140 bytes) from 192.168.1.254:53 to 192.168.1.65:1514 on eth1           │
└ Bottom ────── Elapsed time:   0:07 ──────────────────────────────────────────┘
 Pkts captured (all interfaces):       2282 │ TCP flow rate:       0.60 kbits/s
 Up/Dn/PgUp/PgDn-scroll  M-more TCP info   W-chg actv win  S-sort TCP  X-exit
```

glances

"Glances is a cross-platform system monitoring tool written in Python."



```
xps (Ubuntu 14.04 64bit / Linux 3.13.0-63-generic) - IP 192.168.0.5/24                                          Uptime: 1 day, 17:30:44

Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz - 1.80/1.80GHz   CPU      99.9%  nice:     0.5%  MEM     28.1%  active:    2.47G  SWAP     0.0%  LOAD    4-core
CPU  [||||||||||||||||||||||||||||||||||||||||||| 99.9%]   user:    97.9%  irq:      0.0%  total:  7.71G  inactive:  1.38G  total:  7.91G  1 min:    1.66
MEM  [|||||||||||||                               28.1%]   system:   1.5%  iowait:   0.0%  used:   2.17G  buffers:    199M  used:       0  5 min:    0.98
SWAP [                                             0.0%]   idle:     0.1%  steal:    0.0%  free:   5.54G  cached:    1.76G  free:   7.91G  15 min:   0.68

NETWORK     Rx/s   Tx/s   CONTAINERS 2 (served by Docker 1.7.1)
docker0       0b     0b
lo            0b     0b    Name                             Status   CPU%    MEM IOR/s IOW/s  Rx/s  Tx/s Command
_h35fc066     0b     0b    _dbgrafana_grafana_1         Up 16 mins    0.1  16.1M    0b    0b    0b    0b /usr/sbin/grafana-server --config=/etc/grafana/grafa
_h8e6ba8a     0b     0b    _bgrafana_influxdb_1         Up 16 mins    0.1  16.3M    0b    0b    0b    0b /run.sh
wlan0       19Kb  960Kb
                          TASKS 228 (731 thr), 9 run, 219 slp, 0 oth sorted automatically by cpu_percent, flat view
DISK I/O     R/s    W/s
sda1           0      0   Dropbox              RUNNING
sda2           0   374K   Python               RUNNING       CPU: 4.9% | MEM: 0.3%
sda3           0      0
                            CPU%  MEM%  VIRT   RES   PID USER        NI S     TIME+ IOR/s IOW/s Command
FILE SYS    Used  Total     91.0   0.0 7.13M  100K 22818 nicolargo    0 R   0:11.21     0     0 stress --cpu 4 -t 30
/ (sda2)    197G   226G     85.5   0.0 7.13M  100K 22821 nicolargo    0 R   0:10.62     0     0 stress --cpu 4 -t 30
/boot/efi  3.38M   511M     66.6   0.0 7.13M  100K 22820 nicolargo    0 R   0:10.18     0     0 stress --cpu 4 -t 30
                            64.2   0.0 7.13M  100K 22819 nicolargo    0 R   0:10.14     0     0 stress --cpu 4 -t 30
SENSORS                     40.1   4.0 1.43G  313M 10150 nicolargo    0 S   0:53.53     0  294K /usr/bin/perl /usr/bin/shutter
temp1          °C     27    22.1   1.8 2.67G  140M  3745 nicolargo    0 S   0:47.12    5K  157K /home/nicolargo/.dropbox-dist/dropbox-lnx.x86_64-3.8.8/dropbox /new
temp2          °C     29     9.0  11.4 2.49G  897M  6645 nicolargo    0 S  34:29.50     0     0 /usr/lib/firefox/firefox
Physical id    °C     65     4.9   0.3  240M 25.7M  7077 nicolargo    0 R   0:51.84     0     0 python -m glances
Core 0         °C     63     3.8   2.8 1.60G  217M  3219 nicolargo    0 S   3:40.23     0     0 /usr/bin/gnome-shell
Core 1         °C     65     1.5   1.2  427M 93.4M  1987 root         0 S   4:49.57     0     0 /usr/bin/X :0 -background none -verbose -auth /var/run/gdm/auth-for
Battery         %     31     0.6   0.1  401M 7.42M  4128 nicolargo    0 R   0:02.24     0     0 zeitgeist-datahub
                             0.6   0.2  626M 14.6M  3041 nicolargo    0 S   0:02.61     0     0 /usr/lib/x86_64-linux-gnu/bamf/bamfdaemon
                             0.6   0.3  918M 20.2M  2744 root         0 S   0:12.63     0     0 /usr/bin/docker -d --dns 8.8.8.8 --dns 8.8.4.4
                             0.3   0.7 2.07G 53.4M  2072 rabbitmq     0 S   1:05.63     0     0 /usr/lib/erlang/erts-5.10.4/bin/beam.smp -W w -K true -A30 -P 10485
                             0.3   0.2  468M 16.5M  3325 nicolargo   19 S   0:01.60     0    1K /usr/lib/tracker/tracker-miner-fs
                             0.3   0.4 1.04G 34.6M  4299 nicolargo    0 S   0:00.52     0     0 /usr/lib/evolution/3.10/evolution-alarm-notify
                             0.3   0.1  391M 10.1M  3258 nicolargo    0 S   0:01.21     0     0 /usr/lib/telepathy/mission-control-5
                             0.3   0.1  355M 6.24M  3069 nicolargo    0 S   0:08.60     0     0 /usr/bin/ibus-daemon --daemonize --xim
                             0.3   0.0     0     0   173 root         0 S   0:00.72     0     0 kworker/2:2
                             0.3   0.1  341M 4.64M  4133 nicolargo    0 S   0:00.76     0     0 /usr/bin/zeitgeist-daemon
                             0.3   0.0 39.6M 2.66M  3002 nicolargo    0 S   0:04.69     0     0 dbus-daemon --fork --session --address=unix:abstract=/tmp/dbus-tj7X
                             0.3   0.5  913M 39.9M  4010 nicolargo    0 S   0:06.59     0     0 /usr/bin/python /usr/bin/terminator
                             0.3   0.6  474M 46.2M  9079 nicolargo    0 S   0:11.13     0     0 /usr/lib/firefox/plugin-container /usr/lib/flashplugin-installer/li

                          Warning or critical alerts (lasts 3 entries)
                          2015-09-19 14:57:17 (ongoing) - CPU_USER (98.4)
                          2015-09-19 14:54:48 (0:00:20) - CRITICAL on CPU_USER (98.5)
2015-09-19 14:57:28       2015-09-19 14:53:25 (0:00:24) - CRITICAL on CPU_USER (98.5)
```

B Linux directory structure

- / Root of the filesystem.
  - bin Essential command binaries that need to be available in single user mode.
  - boot Boot loader files.
  - dev Device files
  - etc Host-specific system-wide configuration files
    - opt Configuration files for packages in `/opt`.
    - X11 [Optional] Configuration for the X Window system.
    - sgml [Optional] Configuration for SGML.
    - xml [Optional] Configuration for XML.
  - home Users' home directories
  - lib Libraries needed by the binaries in `/bin` and `/sbin`.
  - lib64 64bit libraries needed by the binaries in `/bin` and `/sbin`.
  - media Mount points for removable media.
  - mnt Temporarily mounted filesystems.
  - opt Optional application software packages.
  - proc Virtual filesystem providing process and kernel information as files.
  - root Home directory for the root user.
  - run Run-time variable data (Info about the running system since the last boot).
  - sbin Essential system binaries
  - srv Site-specific data served by this system (i.e. when used as a server).
  - sys System information about devices, drivers, and some kernel features.
  - tmp Temporary files (volatile).
  - usr Most user utilities and applications are here (read-only).
    - bin User commands.
    - include Header files included by C programs.
    - lib Libraries.
    - local Local hierarchy (empty after main installation)
    - sbin Non-vital system binaries.
    - share Architecture-independent data.
    - lib64 [Optional] 64bit libraries.
    - src [Optional] Source code.
  - var Variable files (whose content is expected to change during system runtime.
    - cache Application cache data.
    - lib Variable state information.
    - local Variable data for `/usr/local`.
    - lock Lock files.
    - log Log files and directories.
    - opt Variable data for `/opt`.
    - run Data relevant to running processes.
    - spool Application spool data.
    - tmp Temporary files preserved between system reboots.

For a more detailed look check out the Filesystem Hierarchy Standard site for the official standards documentation. Alternatively, the Linux Programmer's Manual also provides more granular insights.

C Linux Access Groups

Mostly taken from the excellent Arch WIKI (Users & Groups).

C.1 User

Group	Affected files	Purpose
adm		Administration group, commonly used to give read access to protected logs (inc. full read access to journal files).
ftp	`/srv/ftp/`	Access to files served by FTP servers.
games	`/var/games`	Access to some game software.
http	`/srv/http/`	Access to files served by HTTP servers.
log		Access to log files in `/var/log/` created by `syslog-ng`.
rfkill	`/dev/rfkill`	Right to control wireless devices power state (used by `rfkill`).
sys		Right to administer printers in CUPS.
systemd-journal	`/var/log/journal/*`	Can be used to provide read-only access to the systemd logs, as an alternative to `adm` and `wheel`. Otherwise, only user generated messages are displayed.
uucp	`/dev/ttyS[0-9]+`, `/dev/tts/[0-9]+`, `/dev/ttyUSB[0-9]+`, `/dev/ttyACM[0-9]+`, `/dev/rfcomm[0-9]+`	RS-232 serial ports and devices connected to them.
wheel		Administration group, commonly used to give privileges to perform administrative actions. It has full read access to journal files and the right to administer printers in CUPS. Can also be used to give access to the `sudo` and `su` utilities (neither uses it by default).

C.2 System

Group	Affected files	Purpose
audio*	/dev/audio, /dev/snd/*, /dev/rtc0	Direct access to sound hardware, for all sessions. It is still required to make ALSA and OSS work in remote sessions. Also used in JACK (low latency audio) to give users realtime processing permissions.
dbus		used internally by dbus (the GNU message bus system).
disk*	/dev/sd[a-z][1-9]	Access to block devices not affected by other groups such as optical, floppy, and storage.
floppy*	/dev/fd[0-9]	Access to floppy drives.
input*	/dev/input/event[0-9]*, /dev/input/mouse[0-9]*	Access to input devices (introduced in systemd 215).
kmem	/dev/port, /dev/mem, /dev/kmem	
kvm*	/dev/kvm	Access to virtual machines using KVM.
locate	/usr/bin/locate, /var/lib/locate, /var/lib/mlocate, /var/lib/slocate	See Locate.
lp	/dev/lp[0-9]*, /dev/parport[0-9]*	Access to parallel port devices (printers and others).
mail	/usr/bin/mail	
nobody		Unprivileged group.
optical*	/dev/sr[0-9], /dev/sg[0-9]	Access to optical devices such as CD and DVD drives.
proc	/proc/pid/	A group authorized to learn processes information otherwise prohibited by hidepid= mount option of the proc filesystem. The group must be explicitly set with the gid= mount option.
root	/*	Complete system administration and control (root, admin).
scanner*	/var/lock/sane	Access to scanner hardware.
smmsp		sendmail group.
storage*		Access to removable drives such as USB hard drives, flash/jump drives, MP3 players; enables the user to mount storage devices.
tty	/dev/tty, /dev/vcc, /dev/vc, /dev/ptmx	
utmp	/run/utmp, /var/log/btmp, /var/log/wtmp	
video*	/dev/fb/0, /dev/misc/agpgart	Access to video capture devices, 2D/3D hardware acceleration, framebuffer (X can be used without belonging to this group).

* In older systems (prior to systemd) users had to be manually added to these groups to access the corresponding

devices. This has been depreciated in favour of `udev` and marking the devices with a `uaccess` tag and `logind` assigning the permissions to users dynamically via ACLs according to which session is currently active. Some exceptions exist for newer system setups.

D Common Linux exit codes

Code	Description
0	Success
1	Operation not permitted
2	No such file or directory
3	No such process
4	Interrupted system call
5	Input/output error
6	No such device or address
7	Argument list too long
8	Exec format error
9	Bad file descriptor
10	No child processes
11	Resource temporarily unavailable
12	Cannot allocate memory
13	Permission denied
14	Bad address
15	Block device required
16	Device or resource busy
17	File exists
18	Invalid cross-device link
19	No such device
20	Not a directory
21	Is a directory
22	Invalid argument
23	Too many open files in system
24	Too many open files
25	Inappropriate `ioctl` for device
26	Text file busy
27	File too large
28	No space left on device
29	Illegal seek
30	Read-only file system
31	Too many links
32	Broken pipe
33	Numerical argument out of domain
34	Numerical result out of range
35	Resource deadlock avoided
36	File name too long
37	No locks available
38	Function not implemented
39	Directory not empty
40	Too many levels of symbolic links
42	No message of desired type
43	Identifier removed
44	Channel number out of range
45	Level 2 not synchronized
46	Level 3 halted
47	Level 3 reset
48	Link number out of range
49	Protocol driver not attached
50	No CSI structure available
51	Level 2 halted
52	Invalid exchange
53	Invalid request descriptor
54	Exchange full
55	No anode
56	Invalid request code
57	Invalid slot
59	Bad font file format
60	Device not a stream
61	No data available
62	Timer expired
63	Out of streams resources
64	Machine is not on the network
65	Package not installed
66	Object is remote
67	Link has been severed
68	Advertise error
69	Srmount error
70	Communication error on send
71	Protocol error
72	Multihop attempted
73	RFS specific error
74	Bad message
75	Value too large for defined data type
76	Name not unique on network
77	File descriptor in bad state
78	Remote address changed
79	Can not access a needed shared library
80	Accessing a corrupted shared library
81	.lib section in a.out corrupted

Code	Description
82	Attempting to link in too many shared libraries
83	Cannot exec a shared library directly
84	Invalid or incomplete multibyte or wide character
85	Interrupted system call should be restarted
86	Streams pipe error
87	Too many users
88	Socket operation on non-socket
89	Destination address required
90	Message too long
91	Protocol wrong type for socket
92	Protocol not available
93	Protocol not supported
94	Socket type not supported
95	Operation not supported
96	Protocol family not supported
97	Address family not supported by protocol
98	Address already in use
99	Cannot assign requested address
100	Network is down
101	Network is unreachable
102	Network dropped connection on reset
103	Software caused connection abort
104	Connection reset by peer
105	No buffer space available
106	Transport endpoint is already connected
107	Transport endpoint is not connected
108	Cannot send after transport endpoint shutdown
109	Too many references
110	Connection timed out
111	Connection refused
112	Host is down
113	No route to host
114	Operation already in progress
115	Operation now in progress
116	Stale file handle
117	Structure needs cleaning
118	Not a XENIX named type file
119	No XENIX semaphores available
120	Is a named type file
121	Remote I/O error
122	Disk quota exceeded
123	No medium found
125	Operation cancelled
126	Required key not available
127	Key has expired
128	Key has been revoked
129	Key was rejected by service
130	Owner died
131	State not recoverable
132	Operation not possible due to RF-kill
133	Memory page has hardware error

Taken from nixCraft (25 Jan 2020).