In-Chapter Exercise Solutions

Chapter 0 Basic Commands

1. **la -ls**(there is no **la** command in Linux), **more -q file1**(there is no **-q** option to the more command), **lsblk-a**(there has to be a space between the **lsblk** command and its option **-a**).

2. There has to be a space character separating those components.

3. A single command is just one operation or instruction given to the terminal. It performs a specific action and then returns output or completes the task.

When typing multiple commands, you can combine them in various ways, each serving different purposes. The execution of subsequent commands can depend on the outcome of the previous one, or they can be run sequentially regardless of the outcome. Here are the primary methods to combine multiple commands:

Sequential Execution (;):

Behavior: Executes commands in sequence, one after the other, regardless of the success or failure of the previous command.

Example: cd /var/log; ls will change the directory to /var/log and then list its contents, even if the cd command fails.

Logical AND (&&):

Behavior: Executes the next command only if the previous command succeeds (exits with status 0).

Example: mkdir new\_folder && cd new\_folder will only change the directory to new\_folder if the folder creation is successful.

Logical OR (||):

Behavior: Executes the next command only if the previous command fails (exits with a non-zero status).

Example: gcc program.c || echo "Compilation failed" will only display "Compilation failed" if the compilation does not succeed.

Pipelines (|):

Behavior: Uses the output of the previous command as the input to the next command.

Example: ls | grep "test" will list files and then filter this list to only show files containing "test".

Command Grouping (() and {}):

Behavior: Groups commands together to be executed as if they were a single command. Parentheses run the commands in a subshell, while curly braces run them in the current shell context.

Examples:

(cd /tmp; echo "This runs in a subshell") changes directory and echoes a message in a subshell.

{ cd /tmp; echo "This runs in the current shell"; } does the same but in the current shell context.

4. Before you execute the command, you should know exactly what it’s supposed to accomplish, and then after execution, you can examine the results to see if they match what you expected to happen. Sometimes that’s very difficult, or impossible, to determine.

5. No answer required.

6. The general form is- **cat file1 file2 > output\_file**

7. No answer required.

8. No answer required.

9. No answer required.

10. The **whatis** and **man -k** commands are both used to search for information about commands in Unix-like operating systems, such as Raspberry Pi OS, but they serve different purposes and operate in slightly different ways:

**whatis**

Purpose: Displays a one-line description of a command.

Operation: It searches for the command in the manpage database and returns a brief summary of the command's functionality. This summary is taken directly from the NAME section of the manpage.

Example: Running whatis passwd would typically return something like "passwd - update user's authentication tokens(s)", which is a concise description of what the passwd command does.

**man -k**

Purpose: Performs a keyword search across all manpage descriptions.

Operation: It looks for the specified keyword in the descriptions of all manpages and returns a list of all matches. This can include the command in question but also any other commands and programs that mention the keyword.

Example: Running **man -k passwd** might return entries for the **passwd** command itself as well as other related commands and configuration files, such as chpasswd, passwd(5) (for the passwd file format), and possibly others, each accompanied by a brief description.

11. See ICE 10.

12. On a Raspberry Pi 400 running the latest OS at the time of the writing of this book-

Of course bash is available, and is the default shell.

bob@raspberrypi:~ $ **whereis ksh**

ksh: /usr/bin/ksh /usr/share/man/man1/ksh.1.gz

bob@raspberrypi:~ $ **whereis sh**

sh: /usr/bin/sh /usr/share/man/man1/sh.1posix.gz /usr/share/man/man1/sh.1.gz

bob@raspberrypi:~ $ **whereis csh**

csh: /usr/bin/csh /usr/share/man/man1/csh.1.gz

bob@raspberrypi:~ $ **whereis zsh**

zsh: /usr/share/zsh

13. On our Raspberry Pi 400-

bob@raspberrypi:~ $ **who --all**

system boot 1969-12-31 16:00

bob + tty7 2023-10-04 14:53 old 774 (:0)

run-level 5 2023-10-04 14:53

bob - tty1 2023-10-04 14:53 old 1818

pts/0 2023-10-05 11:08 4781 id=ts/0 term=0 exit=0

14. On our Raspberry Pi 400-

bob@raspberrypi:~ $ **hostname -I**

192.168.1.2

bob@raspberrypi:~ $ **ip addr**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

2: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc mq state UP group default qlen 1000

link/ether dc:a6:32:ee:c6:6b brd ff:ff:ff:ff:ff:ff

inet 192.168.1.2/24 brd 192.168.1.255 scope global dynamic noprefixroute eth0

valid\_lft 49347sec preferred\_lft 38547sec

inet6 fe80::78d9:c72e:75e2:82c/64 scope link

valid\_lft forever preferred\_lft forever

3: wlan0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000

link/ether dc:a6:32:ee:c6:6c brd ff:ff:ff:ff:ff:ff

Chapter 1 ZFS

1.1 No answer required.

1.2 /data/data20file

1.3 /data/text1.txt

1.4 The pathnames to the mirrored text files would be the same, /data2/textfilename! You can’t edit them separately. This is the salient difference between ZFS mirrored and traditional operating system mirrored disks. On a traditional system, for example, the text files would be mirrored on drives C: and D:, so you could designate different pathnames to the two mirrored files. Not so in ZFS.

1.5 No, you can’t mix mirrored pairs with single disks.

1.6 No, as can be seen by what disk usages show in the **zpool iostat -v** command in steps 21. and 22.. data2 is a pool with two mirrors.

1.7 No answer required.

1.8 **cd /sender/data/.zfs/snapshot/2**

1.9 Yes, you can use nano to edit a text file created in a ZFS snapshot directory, just like you would with any other text file. ZFS snapshots provide a point-in-time read-only copy of the file system, including all its directories and files. While the snapshot itself is read-only, you can still access its contents, including text files, and edit them using any text editor, including nano. However, keep in mind that any changes made to files within a ZFS snapshot directory are not persistent across reboots unless you promote the snapshot to a writable filesystem or clone it. So, while you can edit the text file within the snapshot directory using nano, those changes won't be saved unless you take further action to make them permanent.

1.10 You have to make sure you use **truncate** in twice, to create two disk simulation files. One for the pool named source and one for the pool named target. The rest of the solution is exactly the same as the steps in Example 1.2. with pathnames adjusted appropriately.

1.11 On your boot, or system disk, the one that the system is installed on, you can create the file systems under, which is already mounted. On the second disk, after you have found out the logical device name of the disk, you can create the target pool on it with **zpool create**. You can also create destination file system on that pool. The rest of the solution is exactly the same as the steps in Example 1.2, but with pathnames adjusted for the fact that the pools are on different vdevs.

1.12 No answer required.

1.13 Transferring a ZFS pool from one machine to another while maintaining the integrity of the file systems involves several steps. Here's a general outline of the process:

a. Prepare the Destination Machine:

Ensure that the destination machine has the necessary hardware requirements to support the ZFS pool.

Install the ZFS utilities if they are not already installed.

b. Export the ZFS Pool on the Source Machine:

Make sure there are no active processes or operations accessing the ZFS pool.

Export the ZFS pool on the source machine using the **zpool export** command. This ensures that the pool is cleanly dismounted and all data is flushed to disk.

c. Transfer the ZFS Pool:

Transfer the disks or storage devices containing the ZFS pool to the destination machine.

Ensure that the destination machine recognizes the disks. You may need to check disk identifiers to ensure proper identification.

d. Import the ZFS Pool on the Destination Machine:

Once the disks are connected to the destination machine, import the ZFS pool using the **zpool import** command.

Verify that the pool has been imported successfully and that all file systems are intact.

e. Verify Data Integrity:

After importing the pool, it's essential to verify the integrity of the data. ZFS provides built-in mechanisms for data integrity through checksums. You can use commands like **zpool status** and **zfs scrub** to ensure data integrity and repair any errors if necessary.

f. Update Configuration (if necessary):

If the hardware configuration of the destination machine is different from the source machine, you may need to update any relevant configurations, such as network settings or mount points.

g. Test Access and Functionality:

Test access to the file systems on the ZFS pool to ensure that everything is functioning correctly on the destination machine.

Verify that any applications or services relying on the ZFS pool are working as expected.

By following these above steps, you can safely transfer a ZFS pool from one machine to another while maintaining the integrity of the file systems within the pool. Remember to back up critical data before performing any operations to minimize the risk of data loss.

1.14 No answer required, but you should follow the unsharing protocols shown in sub-section IX.

1.15 To implement the three levels of RAID-Z (RAID-Z1, RAID-Z2, and RAID-Z3) in a ZFS pool, you would need a minimum number of disks as follows:

a. RAID-Z1 (Single Parity):

RAID-Z1 requires a minimum of three disks.

It uses one disk for parity, so you need a minimum of two data disks.

b. RAID-Z2 (Double Parity):

RAID-Z2 requires a minimum of four disks.

It uses two disks for parity, so you need a minimum of two data disks.

c. RAID-Z3 (Triple Parity):

RAID-Z3 requires a minimum of five disks.

It uses three disks for parity, so you need a minimum of two data disks.

These are the minimum requirements for each RAID-Z level. However, it's important to note that using more disks can improve performance and fault tolerance. Additionally, always consider the trade-offs between capacity, redundancy, and performance when designing your ZFS storage solution.

1.16 RAID-Z1 (Single parity)

1.17 After executing the provided command correctly,

**zfs send -R -i $snapshot\_yesterday $snapshot\_today | ssh root@$host zfs receive -F $destination**

if you're creating daily snapshots and sending incremental changes from yesterday's snapshot to today's snapshot, the destination machine will have a sequence of snapshots corresponding to each day's changes.

Here's the sequence of events:

Day 1: The initial snapshot is taken on the source machine.

Day 2: Incremental changes between the snapshot from Day 1 and Day 2 are sent to the destination machine, where they are applied.

Day 3: Incremental changes between the snapshot from Day 2 and Day 3 are sent to the destination machine, where they are applied.

So, after three days, you will have snapshots for Day 1, Day 2, and Day 3 on the destination machine, assuming no snapshots are destroyed manually.

Chapter 2 X Window System

2.1 Yes, you would want to switch between GUI and CUI possibly because of hardware resource difficulties, caused by malfunctioning components like external media that the system is running from. Or from misbehaving software apps. To achieve this, you use the systemd **isolate** command.

2.2 If you’re running the Xwayland front end, it’s Mutter. If you’re running Wayland, it’s Wayfire.

2.3 On Bullseye-based Raspberry Pi OS, it’s Mutter.

2.4 Yes, it would be advantageous for the X Window System server to queue requests as well. Here's why:

a. Improved Responsiveness: Queuing requests allows the server to handle them in the order they were received, ensuring fairness and responsiveness, especially in scenarios where multiple clients are sending requests simultaneously.

b. Efficient Resource Utilization: By queuing requests, the server can manage its resources more efficiently. It can prioritize tasks, optimize resource allocation, and prevent overload situations by controlling the rate at which requests are processed.

c. Synchronization: Queuing requests helps in synchronization, ensuring that operations are executed in a deterministic manner. This is crucial for maintaining consistency in the graphical interface, avoiding race conditions, and ensuring that updates are applied in the correct sequence.

d. Error Handling: A queue allows the server to handle errors more gracefully. It can detect and respond to errors in a structured manner, possibly by retrying failed requests, notifying clients of errors, or taking corrective actions without disrupting other ongoing operations.

e. Backpressure Handling: If the server is overwhelmed with requests, queuing provides a mechanism for handling backpressure. Instead of rejecting requests outright, it can buffer them until resources become available, thus preventing potential service disruptions or denial of service situations.

Overall, queuing requests at the server-side complements the event queuing mechanism at the client-side, creating a more robust and efficient communication system within the X Window System architecture.

2.5 Quoting from the text- “Traditionally, on a computer network, a server is thought of as a machine that serves files to many other machines, which is certainly a different function than an X Window System server. In the X Window System, a server is the hardware and/or software that actually takes input from and displays output to the user. For example, the keyboard, mouse, and display screen in front of the user are part of the server; they graphically serve information to the user. The client is an application program that connects to, receives input events from, and makes output requests to the server. Be aware that sometimes (confusingly!) in X Window System jargon, the client is spoken of as a hardware device, like a workstation or computer. We will always use the term client to refer to application program code, rather than to a piece of hardware. In the X Window System, a server and client can exist on the same workstation or computer, and use InterProcess Communication (IPC) mechanisms, such as Linux sockets, to transfer information between them. A local client can be simply thought of as an application that is running on the same machine that you are either sitting in front of, or have a one-to-one relationship with. A remote client is an application that is running on a machine connected to your server via a TCP/IP, or other network connection. Whether a client application is local or remote, it still looks and feels exactly the same to the user of the X Window System.”

2.6 No answer required.

2.7 No answer required.

2.8 No answer required.

2.9 No answer required.

2.10 The X11 library linked in the compiler command for an Xlib program can be either statically or dynamically linked, depending on how the X11 library was built and the options provided during compilation.

If you specify the X11 library using the -lX11 flag in the compiler command, it typically links dynamically by default. This means that the executable will rely on the presence of the X11 dynamic library (libX11.so on Unix-like systems) during runtime.

However, if you want to statically link the X11 library, you would need to provide the path to the static library (libX11.a on Unix-like systems) explicitly in the compiler command. This is usually done by providing the full path to the static library file rather than using the -lX11 flag.

For example, to statically link the X11 library in a compiler command, you might do something like this:

**gcc -o myprogram myprogram.c /usr/lib/libX11.a**

This command explicitly specifies the path to the static X11 library (libX11.a) to be linked with the myprogram executable.

2.11 Initialization to open the display, and create a window. No indeterminate looping. Close the display.

2.12 The XCreateSimpleWindow function in the X Window System is used to create a simple window with basic properties. Here are the arguments it takes:

display: A pointer to the display where the window will be created.

parent: The parent window or the root window where the new window will be placed.

x: The x-coordinate of the window position relative to its parent or root window.

y: The y-coordinate of the window position relative to its parent or root window.

width: The width of the window.

height: The height of the window.

border\_width: The width of the window border.

border: The color of the window border.

background: The color of the window background.

2.13 All three major components, initialization, looping, and closing of the display.

2.14 All three major components, initialization, looping, and closing of the display.

2.15 XsetForeground, XdrawString, gc=XCreateGC, XsetBackground, XfreeGC.

2.16 No answer required.

2.17 In XCB (X C Binding), the function xcb\_create\_window is used to create a new window on the X server. The argument xcb\_connection\_t\* is a pointer to a structure representing the connection to the X server.

The xcb\_connection\_t structure represents the connection to the X server and contains several elements, including:

fd: File descriptor for the connection socket.

proto\_major\_version and proto\_minor\_version: Major and minor version numbers of the X protocol supported by the server.

setup: Pointer to the server's XCB setup information.

screen: Pointer to the default screen of the server.

roots\_len and roots: Number of screens (roots) available and an array of pointers to screen structures.

xcb\_auth\_info\_t: Information related to authentication.

has\_error: A flag indicating whether there has been an error on this connection.

Various other internal elements used for managing the connection and handling events.

These elements allow the XCB library to manage the communication with the X server and handle various aspects of the X protocol. When you call xcb\_create\_window, you pass a pointer to this structure to indicate which X server connection the function should operate on.

2.18 No answer required.

2.19 There’s no indeterminate loop! Particularly the part that exits the loop upon some condition which becomes true. While the cursor is in the window, type **<Ctrl + C>**.

2.20 When you pass screen->root\_visual as an argument to xcb\_create\_window, you're essentially telling the X server to use the default visual of the screen for the new window. This ensures that the new window will have the same pixel format and rendering capabilities as the root window of the screen, which is often desirable for consistency in graphical applications.

2.21 xcb\_create\_gc (c, foreground, win, mask, values); win = screen->root; and values[0] = screen→white\_pixel;

2.22 xcb\_rectangle\_t rectangles[] = {

{ 10, 50, 40, 20},

{ 80, 50, 10, 40}};

2.23 No answer required.

2.24 No answer required.

2.25 No answer required.

2.26

a. Create a file in nano with the 20 data points in it, and save it as something like “linear\_data.txt”.

b. Create the gnuplot script file as follows:

**# Set terminal to XCB to use the X Window System for output**

**set terminal xcb**

**# Set the titles and labels**

**set title "Sample Data Plot"**

**set xlabel "X-axis"**

**set ylabel "Y-axis"**

**# Plot the data**

**plot "linear\_data.dat" using 1:2 with linespoints title "Linear Growth", \**

**"linear\_data.dat" using 1:2:(0.5) with circles title "Data Points"**

c. Save the gnuplot script with a .gp extension, for example, **plot\_data.gp**.

Then, open a terminal or command prompt. Navigate to the directory where you saved plot\_data.gp.

Run the script with gnuplot by typing **gnuplot plot\_data.gp** and pressing Enter.

2.27 This is most easily achieved in gnuplot as follows:

**# Set terminal to persist the plot window**

**set terminal qt persist**

**# Set the range of x values**

**set xrange [-pi:pi]**

**# Increase the sample rate for smoother curves**

**set samples 1000**

**# Label the axes**

**set xlabel "x"**

**set ylabel "y"**

**# Add a title**

**set title "Plot of 2tan(x), sin(x), cos(x), and tanh(x)"**

**# Plot the functions**

**plot 2\*tan(x) title '2tan(x)', \**

**sin(x) title 'sin(x)', \**

**cos(x) title 'cos(x)', \**

**tanh(x) title 'tanh(x)'**

Save the file with a .gp extension, for example, functions\_plot.gp. This extension is not required by gnuplot but helps to identify the file type easily.

Run the script in gnuplot. Open a terminal or command prompt, navigate to the directory where you saved your .gp file, and run the script with gnuplot. Use the following command:

**gnuplot functions\_plot.gp**

2.28 No answer required.

Chapter 3 GNU emacs

3.1 No answer required.

3.2 On our Raspberry Pi system, we are immediately placed in the Welcome Screen buffer only. On the Welcome Screen, make the choice To start… Open a File. Then designate a file name, and perhaps the directory you want it to reside in, and finally make the OK button choice.

3.3 No answer required.

3.4 On our Raspberry Pi system, the buffer into the new file, scratch, and Messages, and async-native-compile-log . We found this out using the pull-down menu choice Buffers. You shift between buffers by using the Buffers menu choices after the listing of the open buffers.

Make the pull-down menu choice File > New Frame. You close it by making the pull-down menu choice File > Delete Frame.

3.5 Now is the time for all good me

Move to another line, and use the paste choice.

3.6 The green (or on our Raspberry Pi system, a light gray) highlighted portion of text in a buffer.

3.7 No answer required.

3.8 No answer required.

3.9 a. Make sure you use the **unalias** command to delete all previously defined aliases.

b. **mv .bashrc .bashrc\_backup**

c. **mv alien .bashrc**

d. Close the terminal window, and open it again.

e. Test the aliases!

f. To clean up, **rm .bashrc**

g. **mv .bashrc\_backup .bashrc**

3.10 No answer required.

3.11 Each of the following solutions assumes that you know ahead of time how many words are on each line and how many lines are in the file you want to use the macro on. They also assume that the cursor is at the beginning of the buffer you want to use the macro on.

a. Delete every other word in a line of unspecified length

Place the cursor at the beginning of the line.

Start recording the macro:

Press F3 or C-x ( (Control-x left parenthesis) to start recording the macro.

Execute the commands to delete every other word:

Move forward a word: M-f (Meta-f or Alt-f).

Delete the next word: M-d (Meta-d or Alt-d).

Move forward a word again: M-f.

Stop recording the macro:

Press F4 or C-x ) (Control-x right parenthesis) to stop recording.

Apply the macro repeatedly across the line:

Press F4 (or the key you've assigned the macro to) repeatedly until you reach the end of the line.

b. Delete every other line in a file of unspecified length

Place the cursor at the beginning of the file.

Start recording the macro:

Press F3 or C-x ( to start recording.

Execute the commands to delete every other line:

Move down one line: C-n (Control-n).

Kill the line: C-k (Control-k). This command kills the text from the cursor to the

end of the line. Since we want to delete the whole line, you might need to press C-k

twice if there's no newline at the end of the line.

Move down one line again: C-n.

Stop recording the macro:

Press F4 or C-x ) to stop recording.

Apply the macro repeatedly across the file:

Keep pressing F4 until you reach the end of the file.

c. Delete every other word and every other line in a file of unspecified length

with lines of unspecified length

This combines the steps from both (a) and (b), and due to the complexity, it might be

more efficient to handle each line individually first for every other word deletion, and

then proceed with every other line deletion throughout the file.

First Part: Delete every other word in each line

Follow the steps in (a) but apply it to each line. You may need to write a more

complex macro or use a combination of macros and manual navigation to apply this

across lines of varying lengths efficiently.

Second Part: Delete every other line in the file

After you've processed the file for words as per your requirements in the first part,

proceed with the steps in (b) to delete every other line in the file.

3.12 **Ctrl-X ( Esc X query-replace ENTER everyone ENTER students ENTER SPACE ENTER Ctrl-X )**

3.13 To find out what the show-paren-mode option is in Emacs, you can use the Emacs Help facility by following these keystrokes:

Access the Help facility: Press C-h (Control-h).

Then press v (for "variable") to look up a description of a variable.

Type show-paren-mode and press Enter.

This sequence (C-h v show-paren-mode RET) will display documentation for the show-paren-mode variable, which controls the Show Paren mode in Emacs.

*show-paren-mode is a variable defined in ‘paren.el’.*

*Its value is nil*

*Documentation:*

*Non-nil means Show Paren mode is enabled.*

*In Show Paren mode, matching parentheses are highlighted based on the value of ‘show-paren-style’.*

*This minor mode works in any buffer, and can be enabled with the command ‘M-x show-paren-mode’.*

*You can customize this variable.*

3.14 a)

**clipboard-yank** Insert the clipboard contents, or the last stretch of killed text.

**query-replace** Replace some occurrences of STRING with NEWSTRING.

**flyspell-mode** Toggle on-the-fly spell checking.

**linum-mode** Toggle display of line numbers in the left margin.

**forward-sentence** Move forward to next end of sentence. With argument, repeat.

**mouse-save-then-kill** Set the region according to CLICK; the second time, kill it.

b) <Alt-x> clipboard-yank

<Ctrl-Alt-%> query-replace

none for flyspell-mode

<Alt-x> linum-mode

<Alt-e>

<mouse-3>

3.15 No answer required.