# Practice M2: Working in the Console

\* NOTE: Some of the exercises in this practice guide are unsuitable for execution in WSL or Docker environments. A virtual infrastructure should be used instead.

For this practice, we will need at least one virtual machine with **AlmaLinux OS 9.x** (or **CentOS Stream 9**, **Oracle Linux 9.x**, or **Rocky Linux 9.x**), **openSUSE Leap 15.x,** or **Debian 12.x** (or **Ubuntu Server 22.04/24.04**) installed. Of course, another version of the listed or another distribution can be used, but there can be some differences.

All commands that we are going to use in this practice will be shown in combination with a prompt. This way, it will be easier for us to understand which user and on which station we are working.

The next steps will be executed on an **AlmaLinux** machine. If there are any discrepancies between **AlmaLinux** and the other two distribution families, there will be a note stating it clearly.

Please note that for each practice we start with a fresh machine or set of machines.

## Part 1

Start the machine.

On the login prompt enter the user and password you specified during the installation process. For example, **lsauser** for username and for the password – **Parolka2@**. Most of the time we will be working with the non-privileged user, as working with **root** all the time is not considered good practice.

Now that we are logged in, let's check what aliases are defined in our installation:

[lsauser@almalinux ~]$ **alias**

alias egrep='egrep --color=auto'

alias fgrep='fgrep --color=auto'

alias grep='grep --color=auto'

alias l.='ls -d .\* --color=auto'

alias ll='ls -l --color=auto'

alias ls='ls --color=auto'

alias xzegrep='xzegrep --color=auto'

alias xzfgrep='xzfgrep --color=auto'

alias xzgrep='xzgrep --color=auto'

alias zegrep='zegrep --color=auto'

alias zfgrep='zfgrep --color=auto'

alias zgrep='zgrep --color=auto'

[lsauser@almalinux ~]$

We can see that even one of the most popular commands (**ls**) is used through alias. In fact, there is more than one alias for **ls**. Of course, the list of aliases and their content varies amongst distributions and installations.

Let's continue with the environment exploration process:

[lsauser@almalinux ~]$ **set**

BASH=/bin/bash

...

HISTFILE=/home/lsauser/.bash\_history

HISTFILESIZE=1000

HISTSIZE=1000

HOME=/home/lsauser

HOSTNAME=almalinux

HOSTTYPE=x86\_64

IFS=$' \t\n'

...

When executed without parameters **set** returns information about the environment – variables and functions. Again, the information will differ from distribution to distribution and between versions of s distribution.

Similar information can be retrieved by executing the **env** or **printenv** commands.

The **set** command can be used to modify the parameters that are driving the environment. In order to see what parameters are there, we can execute:

[lsauser@almalinux ~]$ **set -o**

allexport off

braceexpand on

...

vi off

xtrace off

[lsauser@almalinux ~]$

We can see the same information but prepared for re-use with:

[lsauser@almalinux ~]$ **set +o**

set +o allexport

set -o braceexpand

...

set +o vi

set +o xtrace

[lsauser@almalinux ~]$

Let's change the flag that controls the amount of information shown during commands execution:

[lsauser@almalinux ~]$ **set -x**

++ printf '\033]0;%s@%s:%s\007' lsauser almalinux '~'

[lsauser@almalinux ~]$

You should note that the minus (-) is used to turn on a flag, while the plus (+) is used to deactivate a flag. Now, we can execute a few more commands:

[lsauser@almalinux ~]$ **pwd**

+ pwd

/home/lsauser

++ printf '\033]0;%s@%s:%s\007' lsauser almalinux '~'

[lsauser@almalinux ~]$ **ls -al**

+ ls --color=auto -al

total 16

drwx------. 2 lsauser lsauser 83 Sep 1 22:09 .

drwxr-xr-x. 3 root root 21 Aug 31 13:13 ..

-rw-------. 1 lsauser lsauser 138 Sep 1 22:19 .bash\_history

-rw-r--r--. 1 lsauser lsauser 18 Apr 18 18:35 .bash\_logout

-rw-r--r--. 1 lsauser lsauser 141 Apr 18 18:35 .bash\_profile

-rw-r--r--. 1 lsauser lsauser 376 Apr 18 18:35 .bashrc

++ printf '\033]0;%s@%s:%s\007' lsauser almalinux '~'

[lsauser@almalinux ~]$

We can see that immediately after each command there is information about what is going to be executed. This way we can see that instead **ls**, in fact something else is being executed – its alias or the statement it specifies.

It is time to deactivate the **xtrace** mode:

[lsauser@almalinux ~]$ **set +x**

+ set +x

[lsauser@almalinux ~]$

Let's continue modifying the environment by adding a few variables:

[lsauser@almalinux ~]$ **MYVAR=Demo**

[lsauser@almalinux ~]$ **echo $MYVAR**

Demo

[lsauser@almalinux ~]$

We know that in order to make a variable accessible to child processes as well, we must use the **export** command. Now, by doing the following, we can prove the above statement. First, prepare the variable:

[lsauser@almalinux ~]$ **export MYVAR2=Demo-Child**

[lsauser@almalinux ~]$ **echo $MYVAR2**

Demo-Child

[lsauser@almalinux ~]$

And then start the child process which for this demo will be a nested (or child) bash process:

[lsauser@almalinux ~]$ **bash**

[lsauser@almalinux ~]$

Hm, nothing changed, at leas on the surface. If you are in doubt if you are in a nested (or child) bash process, you can check the value of the **SHLVL** variable:

[lsauser@jupiter ~]$ **echo $SHLVL**

2

[lsauser@jupiter ~]$

Let’s check the values of both variables:

[lsauser@almalinux ~]$ **echo $MYVAR**

[lsauser@almalinux ~]$ **echo $MYVAR2**

Demo-Child

[lsauser@almalinux ~]$

The first one, that was declared without the **export** modifier, is not accessible in the child process. And the second one, which was declared using the **export** modifier is accessible in the child process. Just as we expected it to be.

Now, let’s modify the variable and check the result:

[lsauser@almalinux ~]$ **MYVAR2=Modified**

[lsauser@almalinux ~]$ **echo $MYVAR2**

Modified

[lsauser@almalinux ~]$

Let’s close the child bash process and return to the parent

[lsauser@almalinux ~]$ **exit**

exit

Should we want, we can check the Shell Level again:

[lsauser@jupiter ~]$ **echo $SHLVL**

1

[lsauser@jupiter ~]$

Now, we check the variable again:

[lsauser@almalinux ~]$ **echo $MYVAR2**

Demo-Child

[lsauser@almalinux ~]$

We will see that the changes are not propagated back to the parent process and the variable is with its original value, before modifying it in the child process.

There are other means for child process environment modification without affecting the parent process environment:

[lsauser@almalinux ~]$ **env MYVAR2=Child-Only PS1="CHILD:$PS1" bash**

CHILD:[lsauser@almalinux ~]$ **echo $MYVAR2**

Child-Only

CHILD:[lsauser@almalinux ~]$

This way, we can overwrite a variable that exists in the parent environment only for the child process, or we can create new if one is not existent in the parent. Again, when the child process is over, the value of the variable in the parent environment will be the initial one:

CHILD:[lsauser@almalinux ~]$ **exit**

exit

[lsauser@almalinux ~]$ **echo $MYVAR2**

Demo-Child

[lsauser@almalinux ~]$

If we execute the **env** command we will see information not only about our current environment, but about the environment that the child processes will inherit:

[lsauser@almalinux ~]$ **env**

XDG\_SESSION\_ID=5

HOSTNAME=almalinux

...

HOME=/home/lsauser

LOGNAME=lsauser

MYVAR2=Demo-Child

...

[lsauser@almalinux ~]$

We can see that the **$MYVAR** variable is not on the list. If we execute **set**, it will be there. The explanation is simple - **$MYVAR** is not exported, so it won't be part of the environment of the child processes.

Should we want to get rid of a variable, we can do it like this:

[lsauser@almalinux ~]$ **echo $MYVAR $MYVAR2**

Demo Demo-Child

[lsauser@almalinux ~]$ **unset MYVAR MYVAR2**

[lsauser@almalinux ~]$ **echo $MYVAR $MYVAR2**

[lsauser@almalinux ~]$

Okay, let’s return to the aliases. Why not create our own alias? Let's do it:

[lsauser@almalinux ~]$ **alias inf='uname -a'**

[lsauser@almalinux ~]$ **inf**

Linux almalinux 4.18.0-372.9.1.el8.x86\_64 #1 SMP Tue May 10 08:57:35 EDT 2022 x86\_64 x86\_64 x86\_64 GNU/Linux

[lsauser@almalinux ~]$

This alias will be available only during our current session. Should we want it to persist, we must add its definition to one of the files that are used to control the environment and that we will get to know later.

If we are not sure what is behind a command, is it an alias or it is something else, then we can use the **type** command:

[lsauser@almalinux ~]$ **type ls**

ls is aliased to `ls --color=auto'

[lsauser@almalinux ~]$

If we find the information as not enough, then we can modify the command to:

[lsauser@almalinux ~]$ **type -a ls**

ls is aliased to `ls --color=auto'

ls is /usr/bin/ls

[lsauser@almalinux ~]$

This way we have all the information about the command in question.

We can ask for information about more than one command simultaneously:

[lsauser@almalinux ~]$ **type cd pwd**

cd is a shell builtin

pwd is a shell builtin

[lsauser@almalinux ~]$

If we are interested how the **bash** sees a command, we can use:

[lsauser@almalinux ~]$ **which ls**

alias ls='ls --color=auto'

/usr/bin/ls

[lsauser@almalinux ~]$

On **openSUSE** and **Debian**/**Ubuntu** even though there is a **ls** alias only the path to the binary will be shown.

We can ask for all matches in the **PATH** variable which depending on the settings may return something like:

[lsauser@almalinux ~]$ **which --all ls**

alias ls='ls --color=auto'

/bin/ls

/usr/bin/ls

[lsauser@almalinux ~]$

Again, in **openSUSE** only the two paths to the binary will be shown. Furthermore, on **Debian**/**Ubuntu** the **--all** option is not supported, instead use **-a** which is supported on all distributions.

The reason for the different behavior is that there is a bash function named **which** under **Red Hat**-based distributions and we know already that functions are executed with priority. One way we can see it is to execute:

[lsauser@almalinux ~]$ **which -a which**

which ()

{

( alias;

eval ${which\_declare} ) | /usr/bin/which --tty-only --read-alias --read-functions --show-tilde --show-dot $@

}

/usr/bin/which

[lsauser@almalinux ~]$

In case, we want to know all files (binary, source, documentation) for a command, we can execute:

[lsauser@almalinux ~]$ **whereis ls**

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

[lsauser@almalinux ~]$

During commands execution the Linux system creates and maintains a dictionary, which we can explore with:

[lsauser@almalinux ~]$ **hash**

hits command

2 /usr/bin/bash

1 /usr/bin/whatis

1 /usr/bin/uname

1 /usr/bin/whereis

5 /usr/bin/env

6 /usr/bin/ls

[lsauser@almalinux ~]$

Now, we can clean up a bit by removing our alias:

[lsauser@almalinux ~]$ **unalias inf**

[lsauser@almalinux ~]$ **inf**

-bash: inf: command not found

[lsauser@almalinux ~]$

## Part 2

Now that we know what types of executable artifacts exist, it is time to explore ways of getting help about them.

For the shell built-in commands, we can use the **help** command. If we execute it without parameters, it will return all built-in commands:

[lsauser@almalinux ~]$ **help**

GNU bash, version 4.4.20(1)-release (x86\_64-redhat-linux-gnu)

These shell commands are defined internally. Type `help' to see this list.

Type `help name' to find out more about the function `name'.

Use `info bash' to find out more about the shell in general.

Use `man -k' or `info' to find out more about commands not in this list.

A star (\*) next to a name means that the command is disabled.

job\_spec [&] history [-c] [-d offset] [n] or history -anrw [filename] or history ->

(( expression )) if COMMANDS; then COMMANDS; [ elif COMMANDS; then COMMANDS; ]... [ el>

. filename [arguments] jobs [-lnprs] [jobspec ...] or jobs -x command [args]

: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill >

...

[lsauser@almalinux ~]$

In order to ask for a command, we must execute it like:

[lsauser@almalinux ~]$ **help cd**

cd: cd [-L|[-P [-e]] [-@]] [dir]

Change the shell working directory.

Change the current directory to DIR. The default DIR is the value of the

HOME shell variable.

...

[lsauser@almalinux ~]$

Most external commands offer integrated help. The ways to ask for this information vary, but typically we can use:

[lsauser@almalinux ~]$ **ls --help**

Usage: ls [OPTION]... [FILE]...

List information about the FILEs (the current directory by default).

Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.

-a, --all do not ignore entries starting with .

-A, --almost-all do not list implied . and ..

--author with -l, print the author of each file

-b, --escape print C-style escapes for nongraphic characters

...

[lsauser@almalinux ~]$

For detailed information about commands, configuration files, functions, etc. we can use the **man** utility. Being a tool like any other, we can ask itself for help about its usage:

[lsauser@almalinux ~]$ **man man**

...

MAN(1) Manual pager utils MAN(1)

NAME

man - an interface to the on-line reference manuals

SYNOPSIS

...

man -k [apropos options] regexp ...

man -K [-w|-W] [-S list] [-i|-I] [--regex] [section] term ...

man -f [whatis options] page ...

...

DESCRIPTION

man is the system's manual pager. Each page argument given to man is normally the name of a program, utility or function.

...

When using **man**, it is good to know at least the following set of keys:

* Key **h** shows help information
* Key **q** either exits the help mode or the **man** utility
* If we want to search for something from the current cursor position onwards, then we can press **/** , then enter the string and then press the **Enter** key. Once in this mode, we can press **n** key to move forward (to the next match in the selected direction) over the matches, or the **N** key to move backward (to the previous match)
* Searching backwards (from bottom to top) works the same, but it is initiated with the **?** key

Explore the information and close the tool.

*Please note that depending on the installation man may not be installed as it is not a mandatory component. If this is the case, then it can be installed by executing this (depending on the distribution):*

* *Debian-based:*

***sudo apt update***

***sudo apt upgrade***

***sudo apt install man-db manpages manpages-posix***

* *Red Hat-based:*

***sudo dnf install man-db man-pages***

* *SUSE-based:*

***sudo zypper install man man-pages man-pages-posix***

* *Ubuntu (in case of minimized installation, otherwise follow the Debian approach):*

***sudo unminimize***

*Once done, we may need to update the database by executing:*

***sudo mandb***

*We will discuss package installation in detail in another module.*

Of course, we can quickly scan man for information, by searching in page title only, with:

[lsauser@almalinux ~]$ **whatis ls**

ls (1) - list directory contents

ls (1p) - list directory contents

[lsauser@almalinux ~]$

This is the same as if we executed the following:

[lsauser@almalinux ~]$ **man -f ls**

ls (1) - list directory contents

ls (1p) - list directory contents

[lsauser@almalinux ~]$

A quick search in man can be done on the command line with:

[lsauser@almalinux ~]$ **man -k passwd**

chpasswd (8) - update passwords in batch mode

gpasswd (1) - administer /etc/group and /etc/gshadow

...

[lsauser@almalinux ~]$

Similar effect can be achieved by using the **apropos** command:

[lsauser@almalinux ~]$ **apropos passwd**

chpasswd (8) - update passwords in batch mode

gpasswd (1) - administer /etc/group and /etc/gshadow

...

[lsauser@almalinux ~]$

Beside **man**, usually there is an alternative and modern help system available – **info**. We can use it the same way:

[lsauser@centos ~]$ **info ls**

[lsauser@centos ~]$

For help inside the tool, we can press the **h** key. The help screen can be closed with the **x** key. No matter where we are in the tool, we can exit with the **q** key.

*Note that if native info page is not found, then the corresponding man page will be shown.*

As we already know, file extensions in the **Linux** world are not mandatory. We all will agree that this could cause confusion. Such situations could be handled with the help of the **file** command:

[lsauser@almalinux ~]$ **file /usr/bin/ls**

/usr/bin/ls: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=305acfa1dea641b3c48e3aa9da8cfb74eb3efa6c, stripped

[lsauser@almalinux ~]$ **file /etc/hostname**

/etc/hostname: ASCII text

[lsauser@almalinux ~]$

Additional information about ownership, location, size, timestamps, etc. can be seen with:

[lsauser@almalinux ~]$ **stat .bash\_history**

File: .bash\_history

Size: 215 Blocks: 8 IO Block: 4096 regular file

Device: fd00h/64768d Inode: 8852857 Links: 1

Access: (0600/-rw-------) Uid: ( 1000/ lsauser) Gid: ( 1000/ lsauser)

Context: unconfined\_u:object\_r:user\_home\_t:s0

Access: 2022-09-07 13:04:45.916847878 +0300

Modify: 2022-09-07 13:04:45.916847878 +0300

Change: 2022-09-07 13:04:45.916847878 +0300

Birth: 2022-09-01 22:09:38.614306530 +0300

[lsauser@almalinux ~]$

It is time to continue our exploration with the file and folder related commands.

Let’s check what files we have in our home folder:

[lsauser@almalinux ~]$ **ls**

[lsauser@almalinux ~]$

Depending on the distribution in use, we may either see an empty result or just one or two entries. This does not reflect the situation correctly as it omits the hidden files *(ones whose names start with a dot)*. Should we want to see them as well, we should execute:

[lsauser@almalinux ~]$ **ls -a**

. .. .bash\_history .bash\_logout .bash\_profile .bashrc

[lsauser@almalinux ~]$

Now, this is something else.

Let's create a few folders in our home folder:

[lsauser@almalinux ~]$ **mkdir week-2**

[lsauser@almalinux ~]$ **mkdir -p week-2/dir-1-1/dir-1-2**

[lsauser@almalinux ~]$ **mkdir -p week-2/dir-2-1/dir-2-2**

[lsauser@almalinux ~]$

By using the **-p** option we can create deeper hierarchies and even if some of the parent folders do not exist, they will be created automatically.

Let's enter the **week-2** folder and check its content:

[lsauser@almalinux ~]$ **cd week-2/**

[lsauser@almalinux week-2]$ **ls -al**

total 0

drwxrwxr-x. 4 lsauser lsauser 36 Sep 7 16:25 .

drwx------. 3 lsauser lsauser 97 Sep 7 16:25 ..

drwxrwxr-x. 3 lsauser lsauser 21 Sep 7 16:25 dir-1-1

drwxrwxr-x. 3 lsauser lsauser 21 Sep 7 16:25 dir-2-1

[lsauser@almalinux week-2]$

Okay, what if we needed to create a bigger or deeper hierarchy? It would be better if there was a way to automate it somehow. The good news is that there is a way:

[lsauser@almalinux week-2]$ **mkdir -p dir-{10..20}/sub-dir-{A,B,C}**

[lsauser@almalinux week-2]$ **ls -al**

total 0

drwxrwxr-x. 15 lsauser lsauser 190 Sep 7 16:27 .

drwx------. 3 lsauser lsauser 97 Sep 7 16:25 ..

drwxrwxr-x. 5 lsauser lsauser 57 Sep 7 16:27 dir-10

drwxrwxr-x. 3 lsauser lsauser 21 Sep 7 16:25 dir-1-1

drwxrwxr-x. 5 lsauser lsauser 57 Sep 7 16:27 dir-11

drwxrwxr-x. 5 lsauser lsauser 57 Sep 7 16:27 dir-12

drwxrwxr-x. 5 lsauser lsauser 57 Sep 7 16:27 dir-13

drwxrwxr-x. 5 lsauser lsauser 57 Sep 7 16:27 dir-14

drwxrwxr-x. 5 lsauser lsauser 57 Sep 7 16:27 dir-15

drwxrwxr-x. 5 lsauser lsauser 57 Sep 7 16:27 dir-16

drwxrwxr-x. 5 lsauser lsauser 57 Sep 7 16:27 dir-17

drwxrwxr-x. 5 lsauser lsauser 57 Sep 7 16:27 dir-18

drwxrwxr-x. 5 lsauser lsauser 57 Sep 7 16:27 dir-19

drwxrwxr-x. 5 lsauser lsauser 57 Sep 7 16:27 dir-20

drwxrwxr-x. 3 lsauser lsauser 21 Sep 7 16:25 dir-2-1

[lsauser@almalinux week-2]$

With constructions like **{start..end}** we can easily define set of consecutive values. In case we want to use a list of not necessarily consecutive values, we can just enlist them. For example, we can use **{Value5,Value35,Value48}**.

In a similar manner we can create files. For this purpose, we will use the **touch** command:

[lsauser@almalinux week-2]$ **touch file-1.txt**

[lsauser@almalinux week-2]$ **touch file-{1..5}-{A,B}.txt**

[lsauser@almalinux week-2]$ **ls -al file\***

-rw-rw-r--. 1 lsauser lsauser 0 Sep 7 16:29 file-1-A.txt

-rw-rw-r--. 1 lsauser lsauser 0 Sep 7 16:29 file-1-B.txt

-rw-rw-r--. 1 lsauser lsauser 0 Sep 7 16:29 file-1.txt

-rw-rw-r--. 1 lsauser lsauser 0 Sep 7 16:29 file-2-A.txt

-rw-rw-r--. 1 lsauser lsauser 0 Sep 7 16:29 file-2-B.txt

-rw-rw-r--. 1 lsauser lsauser 0 Sep 7 16:29 file-3-A.txt

-rw-rw-r--. 1 lsauser lsauser 0 Sep 7 16:29 file-3-B.txt

-rw-rw-r--. 1 lsauser lsauser 0 Sep 7 16:29 file-4-A.txt

-rw-rw-r--. 1 lsauser lsauser 0 Sep 7 16:29 file-4-B.txt

-rw-rw-r--. 1 lsauser lsauser 0 Sep 7 16:29 file-5-A.txt

-rw-rw-r--. 1 lsauser lsauser 0 Sep 7 16:29 file-5-B.txt

[lsauser@almalinux week-2]$

We can delete empty folders with the **rmdir** command. If we try to use the **rmdir** with the set of folders we just created it will return an error, as they are not empty:

[lsauser@almalinux week-2]$ **rmdir dir-10**

rmdir: failed to remove 'dir-10': Directory not empty

[lsauser@almalinux week-2]$

In this case, we can use the **rm** command. It can delete both files and folders. Let's first delete all files:

[lsauser@almalinux week-2]$ **rm file-1.txt**

[lsauser@almalinux week-2]$ **ls -al file\***

-rw-rw-r--. 1 lsauser lsauser 0 Sep 7 16:29 file-1-A.txt

-rw-rw-r--. 1 lsauser lsauser 0 Sep 7 16:29 file-1-B.txt

-rw-rw-r--. 1 lsauser lsauser 0 Sep 7 16:29 file-2-A.txt

-rw-rw-r--. 1 lsauser lsauser 0 Sep 7 16:29 file-2-B.txt

...

[lsauser@almalinux week-2]$ **rm file\***

[lsauser@almalinux week-2]$ **ls -al file\***

ls: cannot access 'file\*': No such file or directory

[lsauser@almalinux week-2]$

As you can see, we can use globing with **rm** the same way we use it with **ls**.

Now, let's move one level up and return to our home folder. It is time to clean up a bit:

[lsauser@almalinux week-2]$ **cd ..**

[lsauser@almalinux ~]$ **ls**

week-2

[lsauser@almalinux ~]$ **rm -r week-2/**

[lsauser@almalinux ~]$ **ls**

[lsauser@almalinux ~]$

Let's experiment a little bit with **rmdir** but first prepare the playground:

[lsauser@almalinux ~]$ **mkdir -p dir-{1..5}/sub-dir-{10,20,30}**

[lsauser@almalinux ~]$ **ls**

dir-1 dir-2 dir-3 dir-4 dir-5

[lsauser@almalinux ~]$

Now, what if we try to delete the **dir-1** folder:

[lsauser@almalinux ~]$ **rmdir dir-1**

rmdir: failed to remove 'dir-1': Directory not empty

[lsauser@almalinux ~]$

Of course, we can use the **rm** command to handle the situation but the **rmdir** will also do the job especially because there are no files in the sub-folders:

[lsauser@almalinux ~]$ **rmdir -vp dir-{1..5}/sub-dir\***

rmdir: removing directory, 'dir-1/sub-dir-10'

rmdir: removing directory, 'dir-1'

rmdir: failed to remove directory 'dir-1': Directory not empty

...

rmdir: failed to remove directory 'dir-5': Directory not empty

rmdir: removing directory, 'dir-5/sub-dir-30'

rmdir: removing directory, 'dir-5'

[lsauser@almalinux ~]$ **ls**

[lsauser@almalinux ~]$

Even though we see multiple errors for not being able to delete the parent folder, once there are no more sub-folders the parent folder is deleted as well.

Let's do one more experiment with multiple files to understand better what’s going on under the hood. Again, prepare the playground:

[lsauser@almalinux ~]$ **touch file{1..15}.txt**

[lsauser@almalinux ~]$ **ls**

file10.txt file12.txt file14.txt file1.txt file3.txt file5.txt file7.txt file9.txt

file11.txt file13.txt file15.txt file2.txt file4.txt file6.txt file8.txt

[lsauser@almalinux ~]$

Now, turn on the **xtrace** flag again and try to list all files that satisfy a rule:

[lsauser@almalinux ~]$ **set -x**

++ printf '\033]0;%s@%s:%s\007' lsauser almalinux '~'

[lsauser@almalinux ~]$ **ls file?.txt**

+ ls --color=auto file1.txt file2.txt file3.txt file4.txt file5.txt file6.txt file7.txt file8.txt file9.txt

file1.txt file2.txt file3.txt file4.txt file5.txt file6.txt file7.txt file8.txt file9.txt

++ printf '\033]0;%s@%s:%s\007' lsauser almalinux '~'

[lsauser@almalinux ~]$

As you can see there are two extra steps done – **first the mask gets expanded and substituted with all the matching files**, and **then the execution order gets processed** (currently the alias is executed). Finally, we see the expected result.

Now, turn off the trace flag and remove the files:

[lsauser@almalinux ~]$ **set +x**

+ set +x

[lsauser@almalinux ~]$ **rm file\*.txt**

[lsauser@almalinux ~]$

We can also copy, rename, and move files. Let's execute a few commands.

First, create a local (in our home folder) copy with the same name of an existing file:

[lsauser@almalinux ~]$ **cp /etc/services .**

[lsauser@almalinux ~]$

Then repeat the action but this time the resulting copy will be with different name

[lsauser@almalinux ~]$ **cp -v /etc/services services-1**

'/etc/services' -> 'services-1'

[lsauser@almalinux ~]$

Now, check what we did:

[lsauser@almalinux ~]$ **ls**

services services-1

[lsauser@almalinux ~]$

We ended up with two files with different names but those are exact copies of the same source file.

For the next few commands, we will continue to use the **-v** switch to better show what is happening.

Now, let's create one more copy, but this time we will use the local file:

[lsauser@almalinux ~]$ **cp -v services my-services**

'services' -> 'my-services'

[lsauser@almalinux ~]$ **ls**

my-services services services-1

[lsauser@almalinux ~]$

Let's rename one of the local files:

[lsauser@almalinux ~]$ **mv -v services orginal-services**

renamed 'services' -> 'orginal-services'

[lsauser@almalinux ~]$ **ls**

my-services orginal-services services-1

[lsauser@almalinux ~]$

We can create shortcuts and alternative names for files and folders. This is done by using the soft and hard links.

Hard links are more or less just additional name entries in the file system that point to an existing object – file or folder, thus increasing their **inode** count. Let's first see current situation (with **inode** number in the front):

[lsauser@almalinux ~]$ **ls -li**

total 2040

9021187 -rw-r--r--. 1 lsauser lsauser 692252 Sep 7 17:12 my-services

9019954 -rw-r--r--. 1 lsauser lsauser 692252 Sep 7 16:52 orginal-services

9021186 -rw-r--r--. 1 lsauser lsauser 692252 Sep 7 16:52 services-1

[lsauser@almalinux ~]$

Now create one hard link and examine again:

[lsauser@almalinux ~]$ **ln services-1 services**

[lsauser@almalinux ~]$ **ls -li**

total 2720

9021187 -rw-r--r--. 1 lsauser lsauser 692252 Sep 7 17:12 my-services

9019954 -rw-r--r--. 1 lsauser lsauser 692252 Sep 7 16:52 orginal-services

***9021186 -rw-r--r--. 2 lsauser lsauser 692252 Sep 7 16:52 services***

***9021186 -rw-r--r--. 2 lsauser lsauser 692252 Sep 7 16:52 services-1***

[lsauser@almalinux ~]$

As we can see both files (**services** and **services-1**) have the same **inode** number. This effectively means that they are one and same file, registered under two different names.

Let's delete the original file (**services-1**) and examine the result again:

[lsauser@almalinux ~]$ **rm services-1**

[lsauser@almalinux ~]$ **ls -li**

total 2040

9021187 -rw-r--r--. 1 lsauser lsauser 692252 Sep 7 17:12 my-services

9019954 -rw-r--r--. 1 lsauser lsauser 692252 Sep 7 16:52 orginal-services

***9021186 -rw-r--r--. 1 lsauser lsauser 692252 Sep 7 16:52 services***

[lsauser@almalinux ~]$

The file is still there. In fact, we just deregistered one of the names under which the file is known.

A file is considered deleted when there are no entries (names) pointing to it.

Both the source and the target of a hard link must be on the same file system.

Soft links, in contrary allow the source and the target to be on different file systems.

Let’s create a symbolic or soft link and check the result:

[lsauser@almalinux ~]$ **ln -s /etc/services linked-services**

[lsauser@almalinux ~]$ **ls -li**

total 2040

***9021188 lrwxrwxrwx. 1 lsauser lsauser 13 Sep 7 17:23 linked-services -> /etc/services***

9021187 -rw-r--r--. 1 lsauser lsauser 692252 Sep 7 17:12 my-services

9019954 -rw-r--r--. 1 lsauser lsauser 692252 Sep 7 16:52 orginal-services

9021186 -rw-r--r--. 1 lsauser lsauser 692252 Sep 7 16:52 services

[lsauser@almalinux ~]$

Now compare the **inode** of the linked file to the original one:

[lsauser@almalinux ~]$ **ls -li /etc/services**

***8790802 -rw-r--r--. 1 root root 692252 Aug 31 2020 /etc/services***

[lsauser@almalinux ~]$

They are different. This is due to the linked file being a separate file with its own **inode** number and catalog entry.

## Part 3

As we already know, there is a tool called **sudo**. It is for the situations in which we need administrative privileges for handling a single task. The question that we get asked *(whose password to enter)* varies between distributions.

There are times when we need administrative privileges for a longer period of time. Of course, we can use **sudo** multiple times, but there is also another way. We can switch and act as another user, usually **root**. The command in question is **su**. When using it, we are creating a new session, so we can initiate login or non-login session.

Even though it seems natural to prefer **su** instead of **sudo**, it is not considered a good practice. In general, the recommended way is to use **sudo**.

Let's switch to **root** user by creating a **login** session as **root**:

[lsauser@almalinux ~]$ **su - root**

Password:

Last login: Wed Sep 7 11:26:50 EEST 2022 on tty1

[root@almalinux ~]# **pwd**

/root

[root@almalinux ~]# **ls**

anaconda-ks.cfg

[root@almalinux ~]#

We could omit the word **root** and we do it, the **root** user will be assumed.

*There are distributions, like* ***Ubuntu****, where the* ***root*** *user is locked. For* ***Ubuntu****, in order to unlock it, we must set a password for* ***root****:*

*lsauser@ubuntu:~$* ***sudo passwd root***

*sudo passwd root*

*[sudo] password for lsauser:*

*Enter new UNIX password:*

*Retype new UNIX password:*

*passwd: password updated successfully*

*lsauser@ubuntu:~$*

We can use this situation and compare the results of a few similar commands:

[root@almalinux ~]# **w**

17:42:10 up 6:17, 1 user, load average: 0.01, 0.00, 0.00

USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT

lsauser pts/0 10.0.2.2 12:41 1.00s 0.16s 0.01s sshd: lsauser [priv]

[root@almalinux ~]# **who**

lsauser pts/0 2022-09-07 12:41 (10.0.2.2)

[root@almalinux ~]# **whoami**

root

[root@almalinux ~]# **users**

lsauser

[root@almalinux ~]#

As we can see three of them return that the **lsauser** has a session established and one (the **whoami**) returns a different name – **root**. This is because the **whoami** command returns the **effective user** which is the one as who we act currently. And because we switched to **root**, we are acting as him even though we are logged with the **lsauser**.

We can end our session as the **root** user, by executing:

[root@almalinux ~]# **exit**

logout

[lsauser@almalinux ~]$ **pwd**

/home/lsauser

[lsauser@almalinux ~]$

Most of the next commands will require administrative privileges, and thus they will be prefixed with **sudo**.

Let's create one new user:

[lsauser@almalinux ~]$ **sudo useradd -m -c "New User" newuser**

[sudo] password for lsauser:

[lsauser@almalinux ~]$

Our new user has its own home folder, created in compliance with all system rules. In addition, he has “full name”. Unfortunately, he doesn't have any password yet. Let's deal with this:

[lsauser@almalinux ~]$ **sudo passwd newuser**

Changing password for user newuser.

New password:

**BAD PASSWORD: The password fails the dictionary check - it is based on a dictionary word**

Retype new password:

passwd: all authentication tokens updated successfully.

[lsauser@almalinux ~]$

*The above message usually appears when we try to set up a weak or well-known dictionary word for a password, like* ***Password1****. Obviously, we should not use such passwords.*

The procedure we have followed so far is the same with all distributions. There are distributions that in addition, offer another approach for users' creation. Let's switch to **Debian**/**Ubuntu** and test its approach:

lsauser@ubuntu:~$ **sudo adduser newuser**

Adding user `newuser' ...

Adding new group `newuser' (1001) ...

Adding new user `newuser' (1001) with group `newuser' ...

Creating home directory `/home/newuser' ...

Copying files from `/etc/skel' ...

Enter new UNIX password:

Retype new UNIX password:

passwd: password updated successfully

Changing the user information for newuser

Enter the new value, or press ENTER for the default

Full Name []: New User

Room Number []:

Work Phone []:

Home Phone []:

Other []:

Is the information correct? [Y/n] y

lsauser@ubuntu:~$

We can create groups also:

[lsauser@almalinux ~]$ **sudo groupadd newgroup**

[lsauser@almalinux ~]$

Again, on distributions like **Debian**/**Ubuntu** there is an alternative way:

lsauser@ubuntu:~$ **sudo addgroup newgroup**

[sudo] password for lsauser:

Adding group `newgroup' (GID 1002) ...

Done.

lsauser@ubuntu:~$

We could change the attributes and group membership of the existing users. For example, let's add **newuser** to **newgroup** and check the membership information before and after:

[lsauser@almalinux ~]$ **groups newuser**

newuser : newuser

[lsauser@almalinux ~]$ **sudo usermod -a -G newgroup newuser**

[sudo] password for lsauser:

[lsauser@almalinux ~]$ **groups newuser**

newuser : newuser newgroup

[lsauser@almalinux ~]$

Managing users and groups is just one side of the security equation. Another very important task or set of tasks is to manage their permissions to objects in the file system.

There is one system level value that drives the default permissions for new files and folders. Let's check it:

[lsauser@almalinux ~]$ **umask**

0002

[lsauser@almalinux ~]$ **umask -S**

u=rwx,g=rwx,o=rx

[lsauser@almalinux ~]$ **umask -p -S**

umask -S u=rwx,g=rwx,o=rx

[lsauser@almalinux ~]$

The results may vary amongst distributions. For example, on **openSUSE** the default mask is **0022**, while on **CentOS** and **Ubuntu** it is **0002**.

Now, let's create new file and folder:

[lsauser@almalinux ~]$ **touch new-file**

[lsauser@almalinux ~]$ **mkdir new-dir**

[lsauser@almalinux ~]$ **ls -ald new\***

drwxrwxr-x. 2 lsauser lsauser 6 5 дек 1,37 new-dir

-rw-rw-r--. 1 lsauser lsauser 0 5 дек 1,36 new-file

[lsauser@almalinux ~]$

Let's change the default value of **umask** and try again:

[lsauser@almalinux ~]$ **umask 0022**

[lsauser@almalinux ~]$ **touch new-file-1**

[lsauser@almalinux ~]$ **mkdir new-dir-1**

[lsauser@almalinux ~]$ **ls -ald new\***

drwxrwxr-x. 2 lsauser lsauser 6 5 дек 1,37 new-dir

drwxr-xr-x. 2 lsauser lsauser 6 5 дек 1,41 new-dir-1

-rw-rw-r--. 1 lsauser lsauser 0 5 дек 1,36 new-file

-rw-r--r--. 1 lsauser lsauser 0 5 дек 1,41 new-file-1

[lsauser@almalinux ~]$

We can see that the new objects have different sets of permissions.

If we execute one of the above commands with **sudo**, we can be surprised:

[lsauser@almalinux ~]$ **sudo mkdir new-dir-root**

[sudo] password for lsauser:

[lsauser@almalinux ~]$ **ls -ald new\***

drwxrwxr-x. 2 lsauser lsauser 6 5 дек 1,37 new-dir

drwxr-xr-x. 2 lsauser lsauser 6 5 дек 1,41 new-dir-1

drwxr-xr-x. 2 root root 6 5 дек 1,47 new-dir-root

-rw-rw-r--. 1 lsauser lsauser 0 5 дек 1,36 new-file

-rw-r--r--. 1 lsauser lsauser 0 5 дек 1,41 new-file-1

[lsauser@almalinux ~]$

It appears that the rule for the permissions is the same, but the owner is the **root** user. We can work it out by:

[lsauser@almalinux ~]$ **sudo chown -R lsauser:lsauser new-dir-root**

[lsauser@almalinux ~]$ **ls -ald new\***

drwxrwxr-x. 2 lsauser lsauser 6 5 дек 1,37 new-dir

drwxr-xr-x. 2 lsauser lsauser 6 5 дек 1,41 new-dir-1

drwxr-xr-x. 2 lsauser lsauser 6 5 дек 1,47 new-dir-root

-rw-rw-r--. 1 lsauser lsauser 0 5 дек 1,36 new-file

-rw-r--r--. 1 lsauser lsauser 0 5 дек 1,41 new-file-1

[lsauser@almalinux ~]$

We added the **-R** option in order to apply the changes not only to the folder, but to its child items as well. One more thing – instead of **colon symbol** (**:**) we can use the **dot symbol** (**.**).

The **chown** command allows us to skip one of the two sides – the owner:

[lsauser@almalinux ~]$ **sudo chown lsauser new-dir-root**

Or the group:

[lsauser@almalinux ~]$ **sudo chown :lsauser new-dir-root**

If we must change only the group, then we can go with:

[lsauser@almalinux ~]$ **sudo chgrp newgroup new-dir\***

[lsauser@almalinux ~]$ **ls -ald new\***

drwxrwxr-x. 2 lsauser newgroup 6 5 дек 1,37 new-dir

drwxr-xr-x. 2 lsauser newgroup 6 5 дек 1,41 new-dir-1

drwxr-xr-x. 2 lsauser newgroup 6 5 дек 1,47 new-dir-root

-rw-rw-r--. 1 lsauser lsauser 0 5 дек 1,36 new-file

-rw-r--r--. 1 lsauser lsauser 0 5 дек 1,41 new-file-1

[lsauser@almalinux ~]$

Of course, the permissions can be changed later. For this purpose, we have the **chmod** command. As we already know, it takes arguments in two notations – octal and symbolic. The biggest difference between both methods is that when using octal format, we set all permissions – for owner, group, and others, while with symbolic format we can set, either turn on or off, individual permissions.

Let's set fixed permissions for the **new-dir** folder:

[lsauser@almalinux ~]$ **chmod 665 new-dir**

[lsauser@almalinux ~]$ **ls -ald new\***

drw-rw-r-x. 2 lsauser newgroup 6 5 дек 1,37 new-dir

drwxr-xr-x. 2 lsauser newgroup 6 5 дек 1,41 new-dir-1

drwxr-xr-x. 2 lsauser newgroup 6 5 дек 1,47 new-dir-root

-rw-rw-r--. 1 lsauser lsauser 0 5 дек 1,36 new-file

-rw-r--r--. 1 lsauser lsauser 0 5 дек 1,41 new-file-1

[lsauser@almalinux ~]$ **cd new-dir**

-bash: cd: new-dir: Permission denied

[lsauser@almalinux ~]$

It appears that even if we are the owner of the folder, when we do not have the **execute** permission, we could not change to it. Now, let's return the missing permission, but revoke the one for **read**:

[lsauser@almalinux ~]$ **chmod u-r+x new-dir**

[lsauser@almalinux ~]$ **ls -ald new\***

d-wxrw-r-x. 2 lsauser newgroup 6 5 дек 1,37 new-dir

drwxr-xr-x. 2 lsauser newgroup 6 5 дек 1,41 new-dir-1

drwxr-xr-x. 2 lsauser newgroup 6 5 дек 1,47 new-dir-root

-rw-rw-r--. 1 lsauser lsauser 0 5 дек 1,36 new-file

-rw-r--r--. 1 lsauser lsauser 0 5 дек 1,41 new-file-1

[lsauser@almalinux ~]$ **cd new-dir**

[lsauser@centos new-dir]$ **ls**

ls: cannot open directory .: Permission denied

[lsauser@centos new-dir]$

Now, we are inside the folder, but we could not list its content.

Let's go one step further and copy a file into it, then try to read the file, and the folder:

[lsauser@centos new-dir]$ **cp /etc/os-release .**

[lsauser@centos new-dir]$ **cat os-release**

NAME="AlmaLinux"

VERSION="8.6 (Sky Tiger)"

ID="almalinux"

ID\_LIKE="rhel centos fedora"

VERSION\_ID="8.6"

PLATFORM\_ID="platform:el8"

PRETTY\_NAME="AlmaLinux 8.6 (Sky Tiger)"

ANSI\_COLOR="0;34"

CPE\_NAME="cpe:/o:almalinux:almalinux:8::baseos"

HOME\_URL="https://almalinux.org/"

DOCUMENTATION\_URL="https://wiki.almalinux.org/"

BUG\_REPORT\_URL="https://bugs.almalinux.org/"

ALMALINUX\_MANTISBT\_PROJECT="AlmaLinux-8"

ALMALINUX\_MANTISBT\_PROJECT\_VERSION="8.6"

[lsauser@centos new-dir]$ **sudo ls -al**

total 4

d-wxrw-r-x. 2 lsauser newgroup 24 5 дек 2,19 .

drwx------. 6 lsauser lsauser 267 5 дек 1,47 ..

-rw-r--r--. 1 lsauser lsauser 393 5 дек 2,19 os-release

[lsauser@centos new-dir]$ **ls -al**

ls: cannot open directory .: Permission denied

[lsauser@centos new-dir]$

We should be careful when playing with the permissions as we can end up in funny or strange situations like the one, we saw. It may appear strange, but according to **Linux** it is perfectly fine, this is what we asked, and exactly this we got. ☺

One additional exercise. Let's test one of the special permissions. First, prepare the playground by creating two users and a group, and then make them members of the group:

[lsauser@almalinux ~]$ **sudo useradd user1**

[lsauser@almalinux ~]$ **sudo passwd user1**

...

[lsauser@almalinux ~]$ **sudo useradd user2**

[lsauser@almalinux ~]$ **sudo passwd user2**

...

[lsauser@almalinux ~]$ **sudo groupadd demogroup**

[lsauser@almalinux ~]$ **sudo usermod -aG demogroup user1**

[lsauser@almalinux ~]$ **sudo usermod -aG demogroup user2**

[lsauser@almalinux ~]$

Now, create a folder **/shared** and adjust the group ownership and permissions like:

[lsauser@almalinux ~]$ **sudo mkdir /shared**

[lsauser@almalinux ~]$ **sudo chgrp demogroup /shared**

[lsauser@almalinux ~]$ **ls -ald /shared**

drwxr-xr-x. 2 root demogroup 6 Sep 10 16:44 /shared

[lsauser@almalinux ~]$ **sudo chmod g+w /shared**

[lsauser@almalinux ~]$ **ls -ald /shared**

drwxrwxr-x. 2 root demogroup 6 Sep 10 16:44 /shared

Let's switch to **user1** for example and go to the **/shared** folder and create a file:

[lsauser@almalinux ~]$ **su user1**

Password:

[user1@centos lsauser]$ **cd /shared**

[user1@centos shared]$ **touch fileA.txt**

[user1@centos shared]$ **ls -al fileA.txt**

-rw-rw-r--. 1 user1 user1 0 Sep 10 16:48 fileA.txt

[user1@centos shared]$ **exit**

We can see that the group owner of the file is the user's own group, which will cause some restrictions for **user2**.

Let's modify the permissions of the **/shared** folder and repeat the procedure with new file creation:

[lsauser@almalinux ~]$ **sudo chmod g+s /shared**

[lsauser@almalinux ~]$ **ls -ald /shared**

drwxrwsr-x. 2 root demogroup 23 Sep 10 16:48 /shared

[lsauser@almalinux ~]$

[lsauser@almalinux ~]$ **su user1**

Password:

[user1@centos lsauser]$ **cd /shared**

[user1@centos shared]$ **touch fileB.txt**

[user1@centos shared]$ **ls -al file?.txt**

-rw-rw-r--. 1 user1 user1 0 Sep 10 16:48 fileA.txt

-rw-rw-r--. 1 user1 demogroup 0 Sep 10 16:52 fileB.txt

[user1@centos shared]$ **exit**

Now, all new objects created under this folder will have as their group the group owner of the folder, thus making them shared objects to all members of the group. This is due to the **special group ID** (**SGID**) permission.