# Practice M3: Editors. Files. Streams

\* 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 at 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 virtual machine.

On the login prompt enter the user and password you specified during the installation process. 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.

Let’s create a local copy of the **/etc/passwd** file:

[lsauser@almalinux ~]$ **cp /etc/passwd users.txt**

[lsauser@almalinux ~]$

Usually, the ***standard input*** is linked to the keyboard. In some circumstances we can use a file as a source of input information. Very often we just supply the file name as argument to the command in question:

[lsauser@almalinux ~]$ **cat users.txt**

root:x:0:0:root:/root:/bin/bash

...

lsauser:x:1000:1000:LSA User:/home/lsauser:/bin/bash

[lsauser@almalinux ~]$

We can also use the **stdin** redirection construction:

[lsauser@almalinux ~]$ **cat < users.txt**

root:x:0:0:root:/root:/bin/bash

...

lsauser:x:1000:1000:LSA User:/home/lsauser:/bin/bash

[lsauser@almalinux ~]$

Most often, we want to redirect the ***standard out***, for example to a file:

[lsauser@almalinux ~]$ **ls -l > dir-list.txt**

[lsauser@almalinux ~]$

As a result of the above we will end up with **dir-list.txt** file containing a list of files in the current directory:

[lsauser@almalinux ~]$ **cat dir-list.txt**

total 4

-rw-rw-r--. 1 lsauser lsauser 0 Sep 12 15:13 dir-list.txt

-rw-r--r--. 1 lsauser lsauser 1031 Sep 12 15:11 users.txt

[lsauser@almalinux ~]$

We can see that our target file is also part of the list, but with zero bytes in size. The explanation is simple – **before the command is executed, the redirection is processed**.

If we execute the above command several times, we will notice that each time the file is being recreated, and the existing content is lost. This is not always what we want. We can easily modify the command to:

[lsauser@almalinux ~]$ **ls -l >> dir-list.txt**

[lsauser@almalinux ~]$ **cat dir-list.txt**

total 4

-rw-rw-r--. 1 lsauser lsauser 0 Sep 12 15:13 dir-list.txt

-rw-r--r--. 1 lsauser lsauser 1031 Sep 12 15:11 users.txt

total 8

-rw-rw-r--. 1 lsauser lsauser 127 Sep 12 15:13 dir-list.txt

-rw-r--r--. 1 lsauser lsauser 1031 Sep 12 15:11 users.txt

[lsauser@almalinux ~]$

There is another way of dealing with the constant target file recreation. For this, we must turn on an option in the shell:

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

[lsauser@almalinux ~]$ **ls -l > dir-list.txt**

-bash: dir-list.txt: cannot overwrite existing file

[lsauser@almalinux ~]$

This way we are safe, and we won’t overwrite an existing file by accident.

We can restore the default mode with:

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

We can redirect the error stream as well:

[lsauser@almalinux ~]$ **ls non-existing-file.txt**

ls: cannot access 'non-existing-file.txt': No such file or directory

[lsauser@almalinux ~]$ **ls non-existing-file.txt 2> ls-error.txt**

[lsauser@almalinux ~]$ **cat ls-error.txt**

ls: cannot access 'non-existing-file.txt': No such file or directory

[lsauser@almalinux ~]$

Of course, we can redirect both the **stdout** and **stderr** to different files:

[lsauser@almalinux ~]$ **ls -l ~ /missing > ls-stdout.txt 2> ls-stderr.txt**

[lsauser@almalinux ~]$ **cat ls-stdout.txt**

/home/lsauser:

total 16

-rw-rw-r--. 1 lsauser lsauser 254 Sep 12 15:15 dir-list.txt

-rw-rw-r--. 1 lsauser lsauser 69 Sep 12 15:18 ls-error.txt

-rw-rw-r--. 1 lsauser lsauser 56 Sep 12 15:19 ls-stderr.txt

-rw-rw-r--. 1 lsauser lsauser 0 Sep 12 15:19 ls-stdout.txt

-rw-r--r--. 1 lsauser lsauser 1031 Sep 12 15:11 users.txt

[lsauser@almalinux ~]$ **cat ls-stderr.txt**

ls: cannot access '/missing': No such file or directory

[lsauser@almalinux ~]$

The same way, we can redirect them to one and the same file:

[lsauser@almalinux ~]$ **ls -l ~ /absent > ls-mixed.txt 2>&1**

[lsauser@almalinux ~]$ **cat ls-mixed.txt**

ls: cannot access '/absent': No such file or directory

/home/lsauser:

total 24

-rw-rw-r--. 1 lsauser lsauser 254 Sep 12 15:15 dir-list.txt

-rw-rw-r--. 1 lsauser lsauser 69 Sep 12 15:18 ls-error.txt

-rw-rw-r--. 1 lsauser lsauser 55 Sep 12 15:20 ls-mixed.txt

-rw-rw-r--. 1 lsauser lsauser 56 Sep 12 15:19 ls-stderr.txt

-rw-rw-r--. 1 lsauser lsauser 328 Sep 12 15:19 ls-stdout.txt

-rw-r--r--. 1 lsauser lsauser 1031 Sep 12 15:11 users.txt

[lsauser@almalinux ~]$

We must be aware that the redirection is processed from left to right by the order they are specified. Because of this the result from this command:

[lsauser@almalinux ~]$ **ls -l /absent > ls-out.txt 2>&1**

[lsauser@almalinux ~]$ **cat ls-out.txt**

ls: cannot access '/absent': No such file or directory

[lsauser@almalinux ~]$

*The target file (ls-out.txt) contains the error message.*

Differs from this:

[lsauser@almalinux ~]$ **ls -l /absent 2>&1 > ls-out.txt**

ls: cannot access '/absent': No such file or directory

[lsauser@almalinux ~]$ **cat ls-out.txt**

[lsauser@almalinux ~]$

*The error message is displayed on the screen and the target file is empty.*

New files or documents can be created by utilizing the ***redirection technique*** and the **echo** command:

[lsauser@almalinux ~]$ **echo 'Hello World!' > hello.txt**

[lsauser@almalinux ~]$ **cat hello.txt**

Hello World!

[lsauser@almalinux ~]$

Let’s imagine for a moment that we want to store the current date and time in a file.

We could try to do it with:

[lsauser@almalinux ~]$ **echo "Current date is" date > date.txt**

[lsauser@almalinux ~]$ **cat date.txt**

Current date is date

[lsauser@almalinux ~]$

The result is disappointing. It does not contain what we expected. We must rework our command to:

[lsauser@almalinux ~]$ **echo "Current date is" $(date) > date.txt**

[lsauser@almalinux ~]$ **cat date.txt**

Current date is Mon Sep 12 15:25:02 EEST 2022

[lsauser@almalinux ~]$

Alternatively, we can use:

[lsauser@almalinux ~]$ **echo "Current date is" `date` > date.txt**

[lsauser@almalinux ~]$ **cat date.txt**

Current date is Mon Sep 12 15:25:27 EEST 2022

[lsauser@almalinux ~]$

The substitution can be nested in a text. We must be careful, because there is a difference between double quotes (") and single quotes ('). For example, this will work as expected:

[lsauser@almalinux ~]$ **echo "Current date is `date`" > date.txt**

[lsauser@almalinux ~]$ **cat date.txt**

Current date is Mon Sep 12 15:26:17 EEST 2022

[lsauser@almalinux ~]$

But this one won’t:

[lsauser@almalinux ~]$ **echo 'Current date is `date`' > date.txt**

[lsauser@almalinux ~]$ **cat date.txt**

Current date is `date`

[lsauser@almalinux ~]$

There will be situations in which we would need to create documents on the fly. For this we can use the **heredoc** technique:

[lsauser@almalinux ~]$ **cat << EOF**

> **Here comes the first line**

> **... and then two empty lines**

>

>

> **... this is our final line.**

> **EOF**

Here comes the first line

... and then two empty lines

... this is our final line.

[lsauser@almalinux ~]$

During this exercise we experienced the multi-line prompt defined by the **PS2** variable. Let’s ask for its definition:

[lsauser@almalinux ~]$ **echo $PS2**

>

[lsauser@almalinux ~]$

Of course, we can save **heredoc** to a file:

[lsauser@almalinux ~]$ **cat > simple-file.txt << EOF**

> **Just one line.**

> **EOF**

[lsauser@almalinux ~]$ **cat simple-file.txt**

Just one line.

[lsauser@almalinux ~]$

Let’s create one more file with the help of **echo**, but with some tabs. Write this:

[lsauser@almalinux ~]$ **echo -e 'File with one\tand\t\ttwo tabs.' > file-w-tabs.txt**

[lsauser@almalinux ~]$

Now, let’s use extended and then the plain **cat** syntax to examine the file:

[lsauser@almalinux ~]$ **cat -vET file-w-tabs.txt**

File with one^Iand^I^Itwo tabs.$

[lsauser@almalinux ~]$ **cat file-w-tabs.txt**

File with one and two tabs.

[lsauser@almalinux ~]$

We can compare the two outputs and see that should we need to check the control characters, there is a way.

As we already know, we can execute a series of commands in one iteration. One possible way to accomplish this task is to separate them with semi-colons. This way every command will be executed, no matter how the preceding one ended – successfully or with an error:

[lsauser@almalinux ~]$ **ls > ls-seq.txt ; cat missing-file.txt ; cat ls-seq.txt**

cat: missing-file.txt: No such file or directory

date.txt

dir-list.txt

file-w-tabs.txt

hello.txt

ls-error.txt

ls-mixed.txt

ls-out.txt

ls-seq.txt

ls-stderr.txt

ls-stdout.txt

simple-file.txt

users.txt

[lsauser@almalinux ~]$

Of course, we can rework the command sequence in such a way that every other command will get executed only if the previous completed successfully:

[lsauser@almalinux ~]$ **ls ls\* && ls users.txt && ls missing.txt && echo "Not executed"**

ls-error.txt ls-mixed.txt ls-out.txt ls-seq.txt ls-stderr.txt ls-stdout.txt

users.txt

ls: cannot access 'missing.txt': No such file or directory

[lsauser@almalinux ~]$

*The last command (echo "Not executed") does not get executed because the previous one finished with error*

Alternatively, we can change the behavior so every other command to be executed only if the previous resulted in error:

[lsauser@almalinux ~]$ **ls ls\* || ls users.txt || ls missing.txt || echo "Not executed"**

ls-error.txt ls-mixed.txt ls-out.txt ls-seq.txt ls-stderr.txt ls-stdout.txt

[lsauser@almalinux ~]$

*In fact, only the first command got executed.*

Let’s shorten the sequence:

[lsauser@almalinux ~]$ **ls missing.txt || echo "Executed"**

ls: cannot access 'missing.txt': No such file or directory

Executed

[lsauser@almalinux ~]$

Viable option is to mix the execution modes – in success and on failure:

[lsauser@almalinux ~]$ **ls ls\* && ls missing.txt || echo "Executed"**

ls-error.txt ls-mixed.txt ls-out.txt ls-seq.txt ls-stderr.txt ls-stdout.txt

ls: cannot access 'missing.txt': No such file or directory

Executed

[lsauser@almalinux ~]$

*The* ***echo "Executed"*** *will trigger no matter which of the first two commands is failing*

We can pipe the output of one command to another. We can test it with the **tee** command:

[lsauser@almalinux ~]$ **ls -l | tee tee-out.txt**

total 44

-rw-rw-r--. 1 lsauser lsauser 23 Sep 12 15:26 date.txt

...

-rw-r--r--. 1 lsauser lsauser 1031 Sep 12 15:11 users.txt

[lsauser@almalinux ~]$

Now, let’s do the following to create a small file:

[lsauser@almalinux ~]$ **echo dir1 > new-dirs.txt**

[lsauser@almalinux ~]$ **echo dir2 >> new-dirs.txt**

[lsauser@almalinux ~]$ **echo dir3 >> new-dirs.txt**

[lsauser@almalinux ~]$ **cat new-dirs.txt**

dir1

dir2

dir3

[lsauser@almalinux ~]$

It contains the names of three directories. They do not exist currently.

If we combine the learned so far with the **xargs** command, we can achieve the following:

[lsauser@almalinux ~]$ **cat new-dirs.txt | xargs mkdir**

[lsauser@almalinux ~]$ **ls -ld dir\*/**

drwxrwxr-x. 2 lsauser lsauser 6 Sep 12 15:38 dir1/

drwxrwxr-x. 2 lsauser lsauser 6 Sep 12 15:38 dir2/

drwxrwxr-x. 2 lsauser lsauser 6 Sep 12 15:38 dir3/

[lsauser@almalinux ~]$

This way we can automate things like folder and file creation, users and groups creation, etc.

## Part 2

One quite common operation is to search for a text in either a stream of data or set of one or more files. This can be achieved with the help of the **grep** command. Let’s examine ***how many users in our system are using bash***:

[lsauser@almalinux ~]$ **grep bash users.txt**

root:x:0:0:root:/root:/bin/bash

lsauser:x:1000:1000:LSA User:/home/lsauser:/bin/bash

[lsauser@almalinux ~]$

The same result can be achieved with the help of the ***pipe mechanism***:

[lsauser@almalinux ~]$ **cat users.txt | grep bash**

root:x:0:0:root:/root:/bin/bash

lsauser:x:1000:1000:LSA User:/home/lsauser:/bin/bash

[lsauser@almalinux ~]$

As we can see the result is the same.

Both **cat** and **grep** can process multiple files at once. Let’s experiment with this:

[lsauser@almalinux ~]$ **grep bash users.txt /etc/passwd**

users.txt:root:x:0:0:root:/root:/bin/bash

users.txt:lsauser:x:1000:1000:LSA User:/home/lsauser:/bin/bash

/etc/passwd:root:x:0:0:root:/root:/bin/bash

/etc/passwd:lsauser:x:1000:1000:LSA User:/home/lsauser:/bin/bash

[lsauser@almalinux ~]$

This time, the result differs a bit from this (at least in cosmetic sense):

[lsauser@almalinux ~]$ **cat users.txt /etc/passwd | grep bash**

root:x:0:0:root:/root:/bin/bash

lsauser:x:1000:1000:LSA User:/home/lsauser:/bin/bash

root:x:0:0:root:/root:/bin/bash

lsauser:x:1000:1000:LSA User:/home/lsauser:/bin/bash

[lsauser@almalinux ~]$

We can use **grep** to extract all users whose name starts with **p** *(in fact, all rows that start with)*:

[lsauser@almalinux ~]$ **grep ^p users.txt**

polkitd:x:998:995:User for polkitd:/:/sbin/nologin

[lsauser@almalinux ~]$

Let’s extend our command to return all users whose name starts with either **p** or **s** *(in fact, all rows that start with)*:

[lsauser@almalinux ~]$ **grep ^[ps] users.txt**

sync:x:5:0:sync:/sbin:/bin/sync

shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown

systemd-coredump:x:999:996:systemd Core Dumper:/:/sbin/nologin

systemd-resolve:x:193:193:systemd Resolver:/:/sbin/nologin

polkitd:x:998:995:User for polkitd:/:/sbin/nologin

sssd:x:997:994:User for sssd:/:/sbin/nologin

sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin

[lsauser@almalinux ~]$

In a similar fashion we can ask for those rows that end with **sh**:

[lsauser@almalinux ~]$ **grep sh$ users.txt**

root:x:0:0:root:/root:/bin/bash

lsauser:x:1000:1000:LSA User:/home/lsauser:/bin/bash

[lsauser@almalinux ~]$

Let’s continue our experiments in ***globing*** direction.

We can ask for all ***files which name is four characters long***:

[lsauser@almalinux ~]$ **ls -al /etc/????.conf**

-rw-r--r--. 1 root root 9 Sep 10 2018 /etc/host.conf

-rw-r--r--. 1 root root 812 Aug 25 2021 /etc/krb5.conf

-rw-r-----. 1 root root 1786 Apr 19 01:49 /etc/sudo.conf

-rw-r-----. 1 root tss 7046 Apr 9 2021 /etc/tcsd.conf

[lsauser@almalinux ~]$

We can extend this command to see every matching file’s contents:

[lsauser@almalinux ~]$ **ls /etc/????.conf | xargs cat**

multi on

...

Of course, we can take this expression and use it directly with command like **cat**:

[lsauser@almalinux ~]$ **cat /etc/????.conf**

multi on

...

As we can see the result is the same.

We could extend our expression to match all ***files which name starts with particular letter and has a length of six characters***:

[lsauser@almalinux ~]$ **ls -al /etc/[abc]?????.conf**

-rw-r--r--. 1 root root 1089 Oct 9 2021 /etc/chrony.conf

[lsauser@almalinux ~]$

The condition can be easily reversed – ***all files with name not starting with the provided letters***:

[lsauser@almalinux ~]$ **ls -al /etc/[^abc]?????.conf**

-rw-r--r--. 1 root root 117 Apr 18 21:54 /etc/dracut.conf

-rw-r--r--. 1 root root 19 Aug 31 13:13 /etc/locale.conf

-rw-r--r--. 1 root root 5165 Oct 8 2021 /etc/man\_db.conf

-rw-r--r--. 1 root root 1108 May 10 19:20 /etc/mke2fs.conf

-rw-r--r--. 1 root root 73 Sep 12 15:11 /etc/resolv.conf

-rw-r--r--. 1 root root 449 Apr 18 20:41 /etc/sysctl.conf

[lsauser@almalinux ~]$

Searching for files typically is far more complicated. Usually, we must consider other attributes besides the name.

For this task we can use the **find** command. For example, let’s start with all files named **README** in the **/usr** folder:

[lsauser@almalinux ~]$ **sudo find /usr -type f -name README**

/usr/lib/firmware/rtw88/README

/usr/lib/kbd/unimaps/README

/usr/lib/firewalld/ipsets/README

/usr/lib/dracut/modules.d/10i18n/README

...

We can ask for all files with **conf** extension in the **/etc** folder:

[lsauser@almalinux ~]$ **sudo find /etc -type f -name "\*.conf"**

/etc/resolv.conf

/etc/dnf/dnf.conf

/etc/dnf/plugins/copr.conf

...

Let’s extend the above example by asking for all files with **txt** extension in our system owned by **lsauser**:

[lsauser@almalinux ~]$ **sudo find / -type f -name "\*.txt" -user lsauser**

/home/lsauser/users.txt

/home/lsauser/dir-list.txt

/home/lsauser/ls-error.txt

/home/lsauser/ls-stdout.txt

/home/lsauser/ls-stderr.txt

...

We can reverse the result with:

[lsauser@almalinux ~]$ **sudo find / -type f -name "\*.txt" -not -user lsauser**

/usr/lib/firmware/TDA7706\_OM\_v2.5.1\_boot.txt

/usr/lib/firmware/TDA7706\_OM\_v3.0.2\_boot.txt

/usr/lib/firmware/ar3k/1020200/RamPatch.txt

/usr/lib/firmware/ar3k/1020201/RamPatch.txt

/usr/lib/firmware/ar3k/30000/RamPatch.txt

/usr/lib/firmware/ar3k/30101/RamPatch.txt

...

Or all configuration files that ***changed during the last 24 hours***:

[lsauser@almalinux ~]$ **sudo find /etc -type f -mtime 0 -name "\*.conf"**

/etc/resolv.conf

[lsauser@almalinux ~]$

We can check if this is correct with:

[lsauser@almalinux ~]$ **stat /etc/resolv.conf**

File: /etc/resolv.conf

Size: 73 Blocks: 8 IO Block: 4096 regular file

Device: fd00h/64768d Inode: 8852859 Links: 1

Access: (0644/-rw-r--r--) Uid: ( 0/ root) Gid: ( 0/ root)

Context: system\_u:object\_r:net\_conf\_t:s0

Access: 2022-09-12 15:11:05.358696025 +0300

Modify: 2022-09-12 15:11:04.348677980 +0300

Change: 2022-09-12 15:11:04.356678123 +0300

Birth: 2022-09-12 15:11:04.348677980 +0300

[lsauser@almalinux ~]$

And yes, it should be correct. 😊

We can apply actions on the files found as well:

[lsauser@almalinux ~]$ **sudo find /etc -type f -mtime 0 -name "\*.conf" -exec cat {} \;**

# Generated by NetworkManager

nameserver 192.168.81.1

nameserver 8.8.8.8

[lsauser@almalinux ~]$

Now, let’s continue our journey towards examining what is inside the files.

There is a command like **cat**, but showing the data in reverse and its name is **tac**:

[lsauser@almalinux ~]$ **cat new-dirs.txt**

dir1

dir2

dir3

[lsauser@almalinux ~]$ **tac new-dirs.txt**

dir3

dir2

dir1

[lsauser@almalinux ~]$

*Please note that it* ***does not sort the data in reverse*** *but just displays it from the bottom to the top.*

Should we work with a file that does not fit on the screen, there is a tool. In fact, there are two tools – **less** and **more**:

[lsauser@almalinux ~]$ **less users.txt**

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

adm:x:3:4:adm:/var/adm:/sbin/nologin

lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

sync:x:5:0:sync:/sbin:/bin/sync

shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown

halt:x:7:0:halt:/sbin:/sbin/halt

mail:x:8:12:mail:/var/spool/mail:/sbin/nologin

operator:x:11:0:operator:/root:/sbin/nologin

games:x:12:100:games:/usr/games:/sbin/nologin

ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin

nobody:x:65534:65534:Kernel Overflow User:/:/sbin/nologin

tss:x:59:59:Account used for TPM access:/dev/null:/sbin/nologin

dbus:x:81:81:System message bus:/:/sbin/nologin

systemd-coredump:x:999:996:systemd Core Dumper:/:/sbin/nologin

systemd-resolve:x:193:193:systemd Resolver:/:/sbin/nologin

polkitd:x:998:995:User for polkitd:/:/sbin/nologin

sssd:x:997:994:User for sssd:/:/sbin/nologin

chrony:x:996:993::/var/lib/chrony:/sbin/nologin

sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin

lsauser:x:1000:1000:LSA User:/home/lsauser:/bin/bash

users.txt (END)

We can initiate a search with the **/** key. For example, we can search for **bash**. To go on to the next match, press the **n** key. Quit less with the **q** key.

In order to compare and understand the differences between **more** and **less**, we can experiment with a bigger file, for example **/etc/services** file. First open it with **less**, and then with **more**. Go on and try it.

Relatively often we will need only the first or the last few lines of a file.

For the first, we can use **head**:

[lsauser@almalinux ~]$ **head -n 3 users.txt**

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

[lsauser@almalinux ~]$

And for the last – **tail**:

[lsauser@almalinux ~]$ **tail -n 3 users.txt**

chrony:x:996:993::/var/lib/chrony:/sbin/nologin

sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin

lsauser:x:1000:1000:LSA User:/home/lsauser:/bin/bash

[lsauser@almalinux ~]$

We can use both **head** and **tail** to show all lines but the last or first N (without them).

For example, let’s see all lines but the last 2 (all lines except the last two):

[lsauser@almalinux ~]$ **head -n -2 /etc/passwd**

root:x:0:0:root:/root:/bin/bash

...

chrony:x:996:993::/var/lib/chrony:/sbin/nologin

[lsauser@almalinux ~]$

Let’s add two more rows to the **new-dirs.txt** file for the next few experiments.

[lsauser@almalinux ~]$ **echo dir8 >> new-dirs.txt**

[lsauser@almalinux ~]$ **echo dir5 >> new-dirs.txt**

[lsauser@almalinux ~]$ **cat new-dirs.txt**

dir1

dir2

dir3

dir8

dir5

[lsauser@almalinux ~]$

There are other interesting tools for text manipulation as well.

One such tool is **sort** which does what its name suggests. Let’s execute:

[lsauser@almalinux ~]$ **sort new-dirs.txt**

dir1

dir2

dir3

dir5

dir8

[lsauser@almalinux ~]$

Or in reverse order:

[lsauser@almalinux ~]$ **sort -r new-dirs.txt**

dir8

dir5

dir3

dir2

dir1

[lsauser@almalinux ~]$

With the **nl** tool we can prefix every row with a number:

[lsauser@almalinux ~]$ **nl new-dirs.txt**

1 dir1

2 dir2

3 dir3

4 dir8

5 dir5

[lsauser@almalinux ~]$

We can modify its behavior. For example, we may want to start from 10 with increment of 5 and have three positions for the numbers with a leading zero and a space between the number and the text:

[lsauser@almalinux ~]$ **nl -n'rz' -w3 -s' ' -v100 -i5 new-dirs.txt**

100 dir1

105 dir2

110 dir3

115 dir8

120 dir5

[lsauser@almalinux ~]$

We can count how many users are defined in our system by counting the lines in the **/etc/passwd** file with **wc**:

[lsauser@almalinux ~]$ **wc -l /etc/passwd**

22 /etc/passwd

[lsauser@almalinux ~]$

Should we need information about the shells used in our system, we can use the **cut** command:

[lsauser@almalinux ~]$ **cut -d : -f 7 /etc/passwd**

/bin/bash

/sbin/nologin

/sbin/nologin

/sbin/nologin

/sbin/nologin

/bin/sync

/sbin/shutdown

/sbin/halt

/sbin/nologin

/sbin/nologin

/sbin/nologin

/sbin/nologin

/sbin/nologin

/sbin/nologin

/sbin/nologin

/sbin/nologin

/sbin/nologin

/sbin/nologin

/sbin/nologin

/sbin/nologin

/sbin/nologin

/bin/bash

[lsauser@almalinux ~]$

This list is not easy to read. We must do some post processing. Let’s store it in file and then use the **uniq** command:

[lsauser@almalinux ~]$ **cut -d : -f 7 /etc/passwd > shell-list.txt**

[lsauser@almalinux ~]$ **uniq shell-list.txt**

/bin/bash

/sbin/nologin

/bin/sync

/sbin/shutdown

/sbin/halt

/sbin/nologin

/bin/bash

[lsauser@almalinux ~]$

Okay, now the list is shorter, but still not what we are looking for. And the reason for this is that it is not sorted, so the uniqueness does not work correctly.

Let’s get rid of the intermediate file and assemble the following command:

[lsauser@almalinux ~]$ **cut -d : -f 7 /etc/passwd | sort | uniq**

/bin/bash

/bin/sync

/sbin/halt

/sbin/nologin

/sbin/shutdown

[lsauser@almalinux ~]$

Same result can be achieved with shorter command statement: **cut -d : -f 7 /etc/passwd | sort -u**

Let’s create two more files and experiment with them.

First, create a file with cities:

[lsauser@almalinux ~]$ **cat > f1 << EOF**

> 10:sofia

> 20:athens

> 30:tirana

> 40:rome

> 50:paris

> EOF

[lsauser@almalinux ~]$

Then, another one that will contain countries:

[lsauser@almalinux ~]$ **cat > f2 << EOF**

> 10:bulgaria

> 20:greece

> 30:albania

> 40:italy

> 60:germany

> EOF

[lsauser@almalinux ~]$

Now, let’s paste their content side by side:

[lsauser@almalinux ~]$ **paste f1 f2**

10:sofia 10:bulgaria

20:athens 20:greece

30:tirana 30:albania

40:rome 40:italy

50:paris 60:germany

[lsauser@almalinux ~]$

We can see the result horizontally as well:

[lsauser@almalinux ~]$ **paste -s f1 f2**

10:sofia 20:athens 30:tirana 40:rome 50:paris

10:bulgaria 20:greece 30:albania 40:italy 60:germany

[lsauser@almalinux ~]$

Should we want a different separator, we can use this:

[lsauser@almalinux ~]$ **paste -d : f1 f2**

10:sofia:10:bulgaria

20:athens:20:greece

30:tirana:30:albania

40:rome:40:italy

50:paris:60:germany

[lsauser@almalinux ~]$

Of course, we can link or join the appropriate rows from the files to assemble a result. For this, we should use the **join** command like:

[lsauser@almalinux ~]$ **join -t ':' -j 1 f1 f2**

10:sofia:bulgaria

20:athens:greece

30:tirana:albania

40:rome:italy

[lsauser@almalinux ~]$

The result contains only the rows that have a common key.

We can change the operation to return all from the first file and only that from the second that has a matching key:

[lsauser@almalinux ~]$ **join -t ':' -j 1 -a 1 f1 f2**

10:sofia:bulgaria

20:athens:greece

30:tirana:albania

40:rome:italy

50:paris

[lsauser@almalinux ~]$

Or in the opposite direction:

[lsauser@almalinux ~]$ **join -t ':' -j 1 -a 2 f1 f2**

10:sofia:bulgaria

20:athens:greece

30:tirana:albania

40:rome:italy

60:germany

[lsauser@almalinux ~]$

Or return only these for which there is no correspondence. For the ones from the first file, we should execute:

[lsauser@almalinux ~]$ **join -t ':' -j 1 -v 1 f1 f2**

50:paris

[lsauser@almalinux ~]$

And the same but for the second file:

[lsauser@almalinux ~]$ **join -t ':' -j 1 -v 2 f1 f2**

60:germany

[lsauser@almalinux ~]$

Of course, we can configure the output the way we want. For example:

[lsauser@almalinux ~]$ **join -t ':' -j 1 -o '2.2,1.2' f1 f2**

bulgaria:sofia

greece:athens

albania:tirana

italy:rome

[lsauser@almalinux ~]$

In some situations, we may need to substitute a character or more in a text. For this, we may use the **tr** command.

For example, if we want to delete all zeroes in the sample **f1** file, we can execute:

[lsauser@almalinux ~]$ **cat f1 | tr -d '0'**

1:sofia

2:athens

3:tirana

4:rome

5:paris

[lsauser@almalinux ~]$

In a similar way, we can delete all occurrences of the **a** letter:

[lsauser@almalinux ~]$ **cat f1 | tr -d 'a'**

10:sofi

20:thens

30:tirn

40:rome

50:pris

[lsauser@almalinux ~]$

We can delete more than a single symbol:

[lsauser@almalinux ~]$ **cat f1 | tr -d 'aoie'**

10:sf

20:thns

30:trn

40:rm

50:prs

[lsauser@almalinux ~]$

Similarly, we can change all lowercase occurrences of a letter to uppercase:

[lsauser@almalinux ~]$ **cat f1 | tr a A**

10:sofiA

20:Athens

30:tirAnA

40:rome

50:pAris

[lsauser@almalinux ~]$

Or change all letters to uppercase:

[lsauser@almalinux ~]$ **cat f1 | tr a-z A-Z**

10:SOFIA

20:ATHENS

30:TIRANA

40:ROME

50:PARIS

[lsauser@almalinux ~]$

In the above, we can use classes instead of ranges, but the result will be the same:

[lsauser@almalinux ~]$ **cat f1 | tr [:lower:] [:upper:]**

10:SOFIA

20:ATHENS

30:TIRANA

40:ROME

50:PARIS

[lsauser@almalinux ~]$

A few more examples. Imagine that we have a string, in our case it will be produced with **echo**. And in it, we want to change all consecutive occurrences of a letter to a single one:

[lsauser@almalinux ~]$ **echo 'Multiple aaaaaa and bbbb letters' | tr -s a**

Multiple a and bbbb letters

[lsauser@almalinux ~]$

We can do it for more than one letter:

[lsauser@almalinux ~]$ **echo 'Multiple aaaaaa and bbbb letters' | tr -s ab**

Multiple a and b letters

[lsauser@almalinux ~]$

## Part 3

Let’s first make a copy of the **/etc/services** file locally and open it with **vi**:

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

[lsauser@almalinux ~]$ **vi services**

Now, we can test the navigation and the searching.

Press **G** (the capital letter **G** – **Shift + G**) to go to the end.

Now type in **gg** (two times the lower-case letter **g**) to go to the beginning.

Enter **150G** to go to row **150**. Not sure if this indeed is row **150**? Press **:**, type **set number** and press **Enter**.

Let’s search for a string, for example – **https**. While in command mode type **/https** and press **Enter**. Go to the next match by pressing the **n** key. Repeat it a few times. Now go backward by pressing **N**. Do it a few times.

Close the editor by pressing the **:**, typing **q** and hitting **Enter**.

Now, open the same file, but go directly to a particular row:

[lsauser@almalinux ~]$ **vi +4801 services**

Or to a row, containing a string:

[lsauser@almalinux ~]$ **vi +/VNC services**

Let’s change all occurrences of **tcp** to **TCP**.

First, return to the top (for example, by pressing **gg**).

This can be done by typing **:%s/tcp/TCP/** and pressing **Enter**.

Now if we repeat it once more, there should not be any modification. Let’s do it.

It appears that there are some. Once more?

Again, a few modifications.

The reason for this is that there are rows with multiple occurrences of the word **tcp**.

Now press the **u** key three times to undo the changes.

Correct the search and replace command by typing **:%s/tcp/TCP/g** and pressing **Enter**.

Now all occurrences of **tcp** are replaced with **TCP**.

We can limit the replace action to a range of rows. Let’s add four spaces in front of lines between **100** and **150**.

First go to **100-th** line by typing **100G**. Now type **:100,150s/^/ /** and press **Enter**.

Save the modified file to another file. Type **:w services-modified.txt** and press **Enter**.

Go to line **22** and insert the contents of an external file. For example, insert the **/etc/hostname** file. Type **:r /etc/hostname** and press **Enter**.

In a similar fashion we can insert the result from a command’s execution. Type **:r ! uname -a** and press **Enter**.

Now save the first **100** lines in a separate file by typing **:1,100w services-100.txt** and press **Enter**.

It is time to close the file without saving the changes. Type **:q!** and press **Enter**.

Besides the screen-oriented editors, there are also stream editors such as **sed**.

Let’s create a simple stream and pipe it through **sed**:

[lsauser@almalinux ~]$ **echo "Tram-ta-ra-ra-ram" | sed s/a/A/**

TrAm-ta-ra-ra-ram

[lsauser@almalinux ~]$

If we want all lower-case a turned into upper case, then we must change the command to:

[lsauser@almalinux ~]$ **echo "Tram-ta-ra-ra-ram" | sed s/a/A/g**

TrAm-tA-rA-rA-rAm

[lsauser@almalinux ~]$

We can process files as well:

[lsauser@almalinux ~]$ **sed s/tcp/TCP/g services**

...

systat 11/udp users

daytime 13/TCP

daytime 13/udp

qotd 17/TCP quote

qotd 17/udp quote

...

[lsauser@almalinux ~]$

Of course, the result can be piped through another command like **head** or **tail**. It can also be stored as a file.

We can have more than one processing iteration with:

[lsauser@almalinux ~]$ **sed s/tcp/TCP/g services | sed s/udp/UDP/g**

Of course, the same could be achieved with either this:

[lsauser@almalinux ~]$ **sed 's/tcp/TCP/g ; s/udp/UDP/g' services**

Or this:

[lsauser@almalinux ~]$ **sed -e s/tcp/TCP/g -e s/udp/UDP/g services**

We can return only the changes:

[lsauser@almalinux ~]$ **sed -n s/dns/DNS/pg services**

DNSix 90/tcp # DNSIX Securit Attribute Token Map

DNSix 90/udp # DNSIX Securit Attribute Token Map

sDNSkmp 558/tcp # SDNSKMP

...

Or change only a sub-set of rows:

[lsauser@almalinux ~]$ **sed -n '600,700s/dns/DNS/pg' services**

DNSix 90/tcp # DNSIX Securit Attribute Token Map

DNSix 90/udp # DNSIX Securit Attribute Token Map

[lsauser@almalinux ~]$

We can use **sed** to remove lines matching a pattern as well, for example – ***remove all comments and empty lines***:

[lsauser@almalinux ~]$ **sed '/^#/d ; /^$/d' services**

And save the result in the same file, while retaining a backup copy:

[lsauser@almalinux ~]$ **sed -i.bak '/^#/d ; /^$/d' services**

Now, with the help of our favorite editor 😉, we can open both files side-by-side. We can have them visualized in a split window. As the default installed version is not the full one, let’s use the horizontal split:

[lsauser@almalinux ~]$ **vi -o services.bak services**

We can switch between files with **:n** and **:N** (in the same section) or with (**Ctr+w) + w** (between sections)

If have a default installation of **openSUSE** and **Ubuntu**, we can test the vertical split as well. In order to do it, we must change the switch to **-O** (capital letter O).

To close both files, assuming no changes are made, we can execute **:qa**

Let’s explore a few simple scenarios with the **awk** command.

Should we want the first field of every record, for example, from the **/etc/passwd** file, we can execute:

[lsauser@almalinux ~]$ **cat /etc/passwd | awk -F ':' '{print $1}'**

root

...

lsauser

[lsauser@almalinux ~]$

Or the field that contains the user’s shell:

[lsauser@almalinux ~]$ **cat /etc/passwd | awk -F ':' '{print $7}'**

/bin/bash

/sbin/nologin

...

[lsauser@almalinux ~]$

We can sort and extract just the unique items here with:

[lsauser@almalinux ~]$ **cat /etc/passwd | awk -F ':' '{print $7}' | sort -u**

/bin/bash

/bin/sync

/sbin/halt

/sbin/nologin

/sbin/shutdown

[lsauser@almalinux ~]$

We can apply a filter. For example, return all lines that contain the word **bash**:

[lsauser@almalinux ~]$ **cat /etc/passwd | awk -F ':' '/bash/ {print $1}'**

root

lsauser

[lsauser@almalinux ~]$

Instead of using piping, we can work with the file directly. We can modify the output as well:

[lsauser@almalinux ~]$ **awk -F ':' '{print "user:",$1}' /etc/passwd**

user: root

...

user: lsauser

[lsauser@almalinux ~]$

We can extend the output like:

[lsauser@almalinux ~]$ **awk -F ':' '{print "user:",$1,"shell:",$7}' /etc/passwd**

user: root shell: /bin/bash

...

user: lsauser shell: /bin/bash

[lsauser@almalinux ~]$

We can modify the output even further:

[lsauser@almalinux ~]$ **awk -F ':' '{print "user",$1,"has",$7,"shell"}' /etc/passwd**

user root has /bin/bash shell

...

user lsauser has /bin/bash shell

[lsauser@almalinux ~]$

Let’s create a simple in-line **awk** script to calculate the number of users in the **/etc/passwd** file:

[lsauser@almalinux ~]$ **awk 'BEGIN { FS = ":" } { nlines++ } END { print nlines }' /etc/passwd**

22

[lsauser@almalinux ~]$

We can sort the data while printing it:

[lsauser@almalinux ~]$ **awk 'BEGIN { FS = ":" } { print $1 | "sort" }' /etc/passwd**

adm

bin

...

tss

[lsauser@almalinux ~]$

We can use sort but return just the unique data with:

[lsauser@almalinux ~]$ **awk 'BEGIN { FS = ":" } { print $7 | "sort -u" }' /etc/passwd**

/bin/bash

/bin/sync

/sbin/halt

/sbin/nologin

/sbin/shutdown

[lsauser@almalinux ~]$

**DANGER SECTION BEGIN:**

***The following set of tasks could harm your system. Do them with caution. Feel free to skip the following section.***

Vi is used in a few other tools like **vipw** and **vigr**. With those tools we can change in an easier fashion information about the users, groups, and their passwords, covering all four files.

*Please note that depending on the configuration, commands like* ***vipw*** *may call another text editor instead. This is because they are using the default text editor of the system and not strictly the vi editor*

Let’s rename our user’s login from **lsauser** to **mylsauser** but using NOT this command *(which is the normal case of course not for our or the logged user but for another one)*:

[lsauser@almalinux ~]$ **sudo usermod -l mylsauser lsauser**

But a different and more dangerous approach:

[lsauser@almalinux ~]$ **sudo vipw**

Rename the user (the last line – ***lsauser*** to ***mylsauser***) and save and close the file. You will then see the following reminder:

You have modified /etc/passwd.

You may need to modify /etc/shadow for consistency.

Please use the command 'vipw -s' to do so.

[lsauser@almalinux ~]$

Now, if we try to edit the groups file with (as suggested):

[lsauser@almalinux ~]$ **sudo vipw -s**

We will see that our password is not being accepted anymore, and the reason is that there is no match between our user and a row in the **/etc/shadow** file. If you executed the above, then hit **Ctrl+C** to interrupt the operation.

Our only option is to switch to **root** and make the necessary changes there:

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

Now, let’s change the **/etc/shadow** file by executing:

[root@almalinux ~]# **vipw -s**

Change again the last line (***lsauser*** to ***mylsauser***) and save but instead of just **:wq** enter **:wq!** to force the write

You will see again the following reminder:

You have modified /etc/shadow.

You may need to modify /etc/passwd for consistency.

Please use the command 'vipw' to do so.

[root@almalinux ~]#

Now, rename the main group of the user:

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

Change again the last line (***lsauser*** to ***mylsauser***). In addition, you may need to rename all other instances of the user. For example, if it is a member of the ***wheel*** user group, there we should change ***lsauser*** to ***mylsauser*** as well. Finally, save and quit.

You will see again the following reminder:

You have modified /etc/group.

You may need to modify /etc/gshadow for consistency.

Please use the command 'vigr -s' to do so.

[root@almalinux ~]#

The last step is to edit the **/etc/gshadow** file as suggested:

[root@almalinux ~]# **vigr -s**

Change again the last line (***lsauser*** to ***mylsauser***) and save but instead of just **:wq** enter **:wq!** to force the write

You will see again the following reminder:

You have modified /etc/gshadow.

You may need to modify /etc/group for consistency.

Please use the command 'vigr' to do so.

[root@almalinux ~]#

We should be ready. Exit the root session and return back to our user:

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

logout

[lsauser@almalinux ~]$

Ha, it is still with the old name. It is an expected behavior.

Let’s list the files in our home folder:

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

total 2096

-rw-rw-r--. 1 mylsauser mylsauser 23 Sep 12 15:26 date.txt

drwxrwxr-x. 2 mylsauser mylsauser 6 Sep 12 15:38 dir1

drwxrwxr-x. 2 mylsauser mylsauser 6 Sep 12 15:38 dir2

drwxrwxr-x. 2 mylsauser mylsauser 6 Sep 12 15:38 dir3

-rw-rw-r--. 1 mylsauser mylsauser 254 Sep 12 15:15 dir-list.txt

...

As we can see, the prompt shows the old name, but the file listing shows the new one.

Close the session and reopen a new one, but this time use the new username (***mylsauser***).

**DANGER SECTION END**

***No matter if you executed the above or skipped it, it is safe to continue with the rest of the document.***

***Please note that if you skipped the above section, your prompt will be different (should show lsauser).***

There is a similar tool for modifying the **sudo** allowance – **visudo**. Unlike **vipw** and **vigr**, this tool does syntax check.

We can add one new user named **demouser** with:

[mylsauser@almalinux ~]$ **sudo useradd -m -c "Demo User" demouser**

Do not forget to set a password for the user:

[mylsauser@almalinux ~]$ **sudo passwd demouser**

Changing password for user demouser.

New password:

Retype new password:

passwd: all authentication tokens updated successfully.

[mylsauser@almalinux ~]$

If we want to allow it to execute commands with **sudo**, we have at least two options.

The first one is to add it to a special group. For **Red Hat**-based distributions and **openSUSE** this group is **wheel**:

[mylsauser@almalinux ~]$ **sudo usermod -aG wheel demouser**

And for **Debian** (and its derivatives like **Ubuntu**) this group is **admin** or **sudo**, depending on the version:

[mylsauser@ubuntu ~]$ **sudo usermod -aG sudo demouser**

Another option is to edit a special configuration file **/etc/sudoers**. This can be done with any text editor, or with a special tool – **visudo**.

*Please note that on some distributions, depending on the installation profile, the* ***sudo*** *functionality may not be installed by default. One such case is a minimal* ***Debian****-based installation.*

Let’s do it and open the file for editing:

[mylsauser@almalinux ~]$ **sudo visudo**

and add the user *(****demouser****)* just after the **root** *(around line 100, at least on* ***AlmaLinux****)*. So, the block will become:

## Allow root to run any commands anywhere

root ALL=(ALL) ALL

**demouser ALL=(ALL) ALL**

Save the changes and log out. Now log in back, but this time with the **demouser** and try to execute command with **sudo**.

*Next section is not mandatory and is applicable to* ***SUSE*** *and* ***openSUSE*** *only. Feel free to skip it.*

*Not all distributions are made the same. You can encounter different behaviors. For example, with* ***openSUSE*** *this functionality is organized a little bit differently – all users can act as administrators, but they should use the* ***root*** *password when escalating. We can modify this if we edit the* ***/etc/sudoers*** *file and comment the following two lines:*

***#****Defaults targetpw # ask for the password of the target user i.e. root*

***#****ALL ALL=(ALL) ALL # WARNING! Only use this together with 'Defaults targetpw'!*

*Then we can add a record for the user in question (either the current one –* ***mylsauser*** *or any other) under the one for* ***root****:*

*root ALL=(ALL) ALL*

***lsauser ALL=(ALL) ALL***

*Now, we can save and then log out. Once log in again, we will see that the* ***sudo*** *behavior is the same as with the other two distributions.*

Besides the group membership and changing the **/etc/sudoers** file, there is one more method.

In fact, it is the recommended option. It is to create a file with the appropriate instructions and store it in a special place.

For example, if we had a **demouser2**, and we want to give him/her **sudo** permission, we would create a file that contains:

**demouser2 ALL=(ALL) ALL**

And save it to the **/etc/sudoers.d/demouser2** or another name that is meaningful.

In all the above examples, if we substitute the last **ALL** with **NOPASSWD: ALL** we will allow the user to use **sudo** without the need to enter a password.

For more information about this functionality, explore the **/etc/sudoers** file or check the help by executing:

[mylsauser@almalinux ~]$ **man 5 sudoers**