|  |  |
| --- | --- |
| **DOCUMENT RULES:** | |
| **Task Number / Name:** | **Linux Fundamentals** |
| **Task name & column name should be written:** | **Bold (CTRL+B)** |
| **Commands should be written in the after # sign:** | *Italic (CTRL+I) #hostname* |
| **Output photo should be cropped or compressed:**  **Photo could be more than one:**  **If you need extra lines, add the line next after it:** | ***Description photo should be with title bar (CTRL + I + B)*** |
| **All other text should be written:** | Standard |
| **Font name and text size:** | Calibri and 9 |
| **Group name:** | Dev\_ops\_ |
| **Student name and surname:** | Murad Abbaszade |
| **E-mail:** | [muradabbaszade6@gmail.com](mailto:muradabbaszade6@gmail.com) |
| **WhatsApp number:** | **+994703664205** |

Linux files

cd –change directory

We can list the current directory using pwd

Now we create the file touch my-new-file

echo amazing > /etc/my-second-file amazing sozu fayl icine yazilir.

rm /var/dont-need-this.png remove file

Man

1 Executable programs or shell commands

2 System calls (functions provided by the kernel)

3 Library calls (functions within program libraries)

4 Special files (usually found in /dev)

5 File formats and conventions eg /etc/passwd

6 Games

7 Miscellaneous (including macro packages and conventions), e.g.man(7), groff(7 8 System administration commands (usually only for root)

9 Kernel routines [Non standard

what sections/pages we have available? man -f ls

man 1 ls = see Executable programs or shell commands

This time it will be intro. This command in the introduction to Linux commands. man -f intro

man intro  
man 1 intro  
man 8 intro

man -w ls returns the location of the file from where the page is rendered.

In order to create directory we need to use mkdir command.

mkdir myfirstdirectory creates directory with provided name.

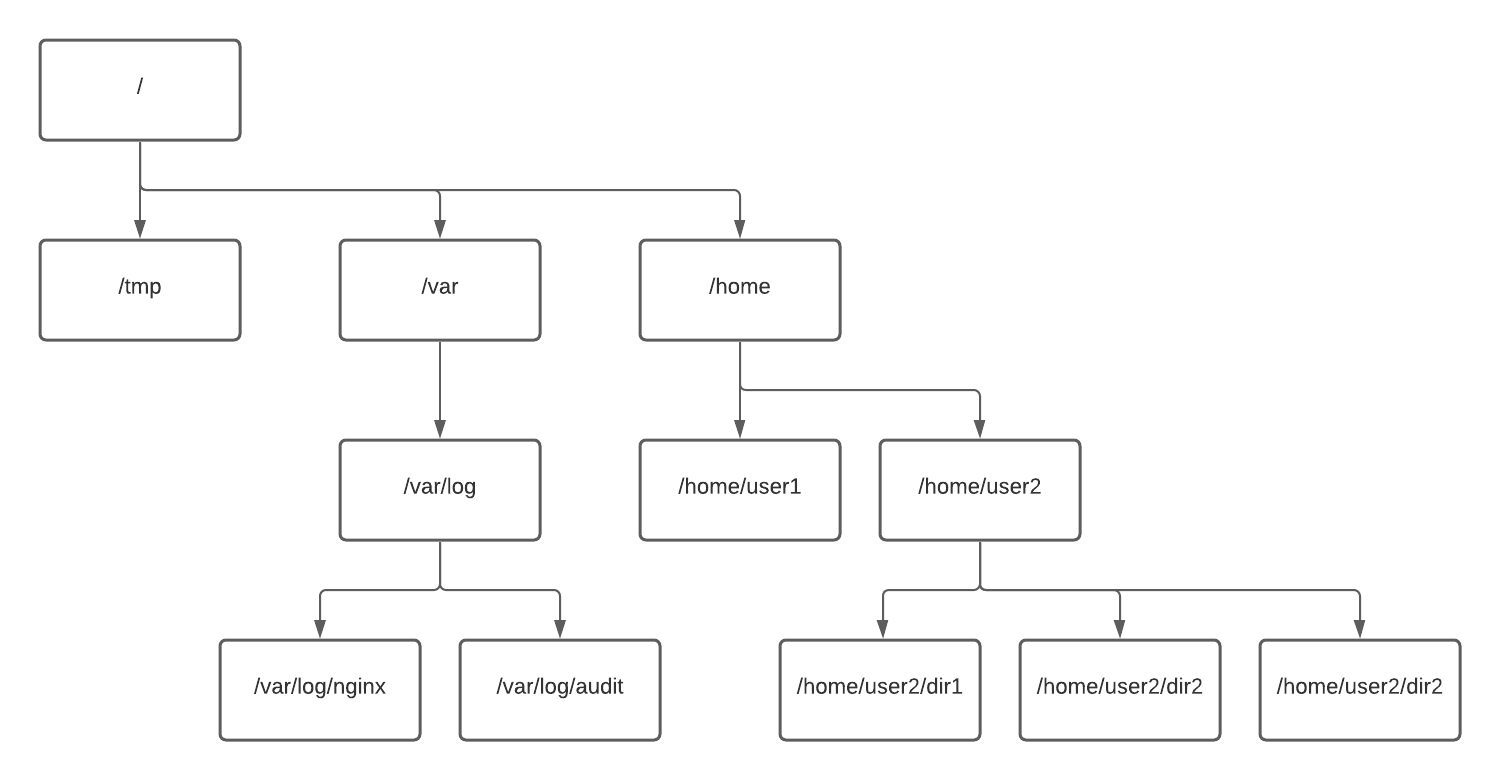
Let's create 10 directories, starting with testdir1 to testdir10. How to do it in **one** command?mkdir testdir{1..10}

ls -l parentdirIn this way we can list files in another directory.

pwd . This command shows your current work directory.

cd ../.. 2 defe geriye qayidir

In Linux filesystems, the root, the "first point" is a root directory. This root directory is represented by /. Everything what is on the top level of the filesystem is in / directory.

Let's take a look at the picture below.

cd /var/log  
pwd  
cd go to /root  
pwd

we will remove the the root directory.rmdir root

rmdir testdir{1..10}

We have some directory called maindir, let's look insidels maindir

rm -rf anotherparentdir

Wow, that was it! But.. It didn't work wit rm earlier.

Now we used some arguments

-r means go recursively through directories (and treat everything as file)

-f - force. Another words, do not ask, assume the user knows what he is doing.

touch my{01..100}file Let's use the wildcard character \* to list all of them ls my\*file

touch try1 try2 try01 Ok, now, we want to list try1 and try2 only. If I use ls try\* we will receive all three files.

Instead of \* we can use ?. This sign means any single character. Let's try ls try?

We can remove single file rm try01 We can remove more than one file rm try1 try2

vim mynewfile

Now, hit ESC key to enter the command mode, and type :wq

You left vim and now you can check the file ls -l mynewfile

THE PIPES

The pipes and redirections are used to send (or retrieve) some information sent from one command or script to another command or script. It works on files too. Let's think about some examples

-Count number of lines in the file

-Select uniq values from one file and write it in another

-Find occurences of some string in the file or system

-and many more

Before we start, here are some commands which we will use when learning pipes. Obviously, all these commands can be used independently.

grep. This command search for given pattern in the output. Output may be the file or output from other command.

grep 'case' .bashrc This command will search for pattern case in a file .bashrc.

wc is a utility for counting words, newlines, bytes. Commonly we use it for counting lines.

When we execute

wc -l .bashrc

it will count how many lines (-l argument) are in our .bashrc file.

sort will sort the output in alphabetical order

sort numbers.txt will sort the prepared file with generated numbers.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

By using uniq we can limit the occurences of the same record to only one. But in order to have this done properly, sort is needed, therefore, we need to pipe these one to another.

Ok, the file is quite huge. How many records we have? Let's pipe the output from the command we just used to wc! cat numbers.txt | wc –l ekrana setirlerin sayi cixir –l(lines).

cat numbers.txt | uniq |wc –l

Hm... something is not right. Let's think...(netice yanlis cixir sebebi odur ki uniq yanliz sort olunan fayllarda duzgun isleyir buna gore de evvelce faylin icindekileri sort etmeliyik sonar uniq istifade etmeliyik)

Did we piped our data correctly? At this moment, we can say - yes. We printed the file, select unique values and count them. But... Did we, really?

Did we missed something? Yes, we did. In this case we have here, we used uniq on unsorted data. That is why we have close to 9000 values, when we should have max 100 (as the script which generated the file used random values between 1 and 100). It is important to remember, uniq always works best with sort. And sort is first.

Let's try.

cat numbers.txt | sort | uniq | wc –l

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Now, let's redirect this to the file:

ls -al > directorylist.txt

What we did here?

ls -al - we know this already

redirects all output from the left side of the sign, to the file on the right side of the sign

directorylist.txt file where the redirected output will be sent.(ls –al , a(all), l(long), > ls –al dan alinan neticeni txt faylina gonderir.)

Well, the > redirection will do as follows:

-If file doesn't exist, create it

-Add content from redirected output

-If file exists and it is not empty, clear the file and write the redirected output in empty file

Ok, so how to *append* and not *rewrite* the file? Is it possible?

Yes, it is. >>

Reading the file

vim text.txt

view text.txt

more

The first command which we can use in order to have better experience is more.

more testfile

It we want to print first lines of the file, we can use head. By default head shows 10 lines.

head testfile

tail does exactly the same thing as heead does, but from the end of the file.

tail testfile prints 10 last lines.

Copy and Move Files

tail -n22 testfile will show 22 lines. tree sourcedir

We used here a new command, tree. It recursively shows the content of the directory:

sourcedir

|-- one

|-- sourcesubdir

| |-- four01

| |-- four02

| |-- four03

| |-- four04

| |-- four05

|-- three01

|-- three02

|-- three03

cp source target

cp two01 two02 ../targetdir (hazirda yerlesiyimiz directory de targetdit directoryisi yoxdur ona gore de arxaya getmek lazimdir. .. two01 ve two02 nin parent folderin icindeki targetdir-e kopyalanmasini gosterir.)

-R argument means *recursively*

mv file1 file2 file3 targetlocation – MOVE

We do it with a new command - diff. This is the way to compare two files.

diff .profile newfilewithcontent.txt – 2 fayl arasindaki ferqlilikleri ekrana cixarir

The Top Command

who The main purpose of who is to show who is logged in.

top - 19:38:28 up 2 days, 20:47, 0 users, load average: 0.52, 0.58, 0.59

In the first line we see something similar to the example above. Let's go through it one by one.

top - program name  
19:38:28 - current hour, obvious :)  
up 2 days, 20:47 - uptime. Another words, the time from last start of the system.  
0 users - number of *active* users. Here we can see similar information, like with command who . Let's try. First, we need to quit the top:

q

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

load average: 0.52, 0.58, 0.59

Let's go through it.

We see here three numbers. They are representing the load average for the system in last 1, 5 and 15 minutes. These shows the average number of processes **running** and **waiting for CPU** time.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Tasks: 6 total, 1 running, 5 sleeping, 0 stopped, 0 zombie

Second line shows us information about processes in our system. What every type means?

total - shows all processes in the system

running - currently active processes. It means, these processes are using CPU right now

sleeping - generally - process is waiting for something. It may be I/O operation for example.

stopped - Stopped processes (for example by ctrl+z)

zombie - Very important state to understand. It is a process which had finish his job but still has entry in the process table. In simple way, these processes are waiting for exit(). It may happen, when parent process deteriorated somehow. Sometimes we are able to kill zombie (by killing the parrent), but in may cases it will not work. But it is not a place to talk about it

The Ps Command

top -o %MEM - run top with processes sorted by memory ps . This way we can see these processes, which are related to this specific session only

PID - quite obvious, this is the process id.

TTY - Terminal associated with the process. [Here](https://www.howtogeek.com/428174/what-is-a-tty-on-linux-and-how-to-use-the-tty-command/) is a very detailed reading about TTY.

TIME - total time of CPU usage

CMD - the command which is running To list all processes, use

ps –A

ps -f -u syslog

shows all processes run by user syslog.

ps -f -C cron

shows all processes, where the executable is cron.

ps -f -p 1

shows process with specified PID.

Alias

ps -f --ppid 1 Aliases give us a great possibility to "shape" command line to our needs.

Let's suppose, we want to list all files in long format and with hidden files:

ls –al

We have to use 6 characters. not much. But if we need to repeat this command again and again, it may become tough. So, what about this:

Ll

In order to create alias we need to use alias command. This time it is not an abbreviation :). Ok, let's create an alias for ls with long list, hidden files and human-readable format of size.

alias lh='ls -alh'(lh yazanda ls –alh kodu isleyecek)

alias shows all defined aliases in the session.

unalias lh Can we remove alias

Users\*

The whoami command shows the user we are logged in.

we can use this tool to ask about another user: id ubuntu

cat /etc/shadow

The part of the file, which we want to understand now are the first and second element. It is a username and password.

The file /etc/group contains information about groups.

The last file is /etc/gshadow.

clear && cat /etc/gshadow

This file contains encrypted passwords for groups.

useradd

useradd testuser1

adduser

adduser testuser2

Comparison

First difference is that adduser is more verbose. We see what is happening.

Second, very significant difference is that adduser is interactive. It asks for password for user and other information. Provide some test data during the execution of the binary. Let's have something to compare.

Let's see all accounts now.

grep testuser /etc/passwd

grep testuser /etc/shadow

ls -l /home

passwd testuser3 change password

History

Ok, nice. But if you already typed 1000 commands and you want to go back to the command number 268 it will be... not very convenient. In this case we can call the history:

history(onceden istifade olunmus kodlari qaytarir)

Now, you can simply copy/paste the command, but it is not very professional

history 3

will show 3 last commands stored in history.

history –c clear history

Elevate Privileges\*

Root

Root is a master of the system. Root can do everything. In order to use another user we need to use command su. su has multiple options. You can learn them with

su –h

Let's use it to switch to student1.

su - student1

Ok, let's go back to the initial user - root.

Exit

When su log us as root, sudo elevates our privileges on behalf of root

Work with logs

Let's navigate to default log directory and take a look.

cd /var/log

ls -al

|  |  |
| --- | --- |
| **Log name** | **Purpose** |
| syslog | The main system log. Contains all important information about the system and applications. Generally, if something is not writting its own logs, it will be here, in syslog. |
| auth.log | Contains information about authorizations. All user login attempts (with information if successful or not), logout, password changes, remote logins and use of sudo. |
| dmesg | is a kernel ring buffer, not the log (as we understand the logs). It allows us to interact with kernel and get information by querying bootup messages. It doesn't mean, that dmesg contains the booting data, but everything what is going on during the system's work. |
| kern.log | Stores Kernel messages |
| boot.log | Contains system' starting sequence (not kernel boot). Another words, information about started services, applications, disks configurations and so on. |
| lastlog | file to be used with lastlog utility. Contains information abount last logins |
| faillog | similar to lastlog, use this file with faillog utility. Logs fails, like login failures |
| wtmp.log | Contains login infomration. However, it doesn't show information similar to lastlog, but used by other utilities, like who. |
| dpkg.log | Contains data about packages management - install, remove, update, etc. |

ls -al /var/log/nginx

Here we have nginx directory with its logs.

syslog message levels

0: emerg

1: alert

2: crit

3: err

4: warning

5: notice

6: info

7: debug

Cron

Let's see a few ways how we can check the status of the service.

systemctl status cron.service

ps aux | grep cron

Shows processes related to the cron service. We should see something similar to

root 565 0.0 0.1 8536 3084 ? Ss

cron is a service which controls multiple crontabs

crontab is simply a list of tasks or commands which are scheduled to be executed on specific date and time.

ls -al /etc/cron.\* -d These directories contain scripts which are executed... daily, hourly, monthly or weekly.

We have four anacron command executed with different scheduling. We will learn about it later. What is anacron?

It is scheduler too. But works a little bit differently than cron. Unlike the cron, anacron doesn't assume that the machine is run continuously without stops. It is crucial for system's scripts which must be executed daily, for example.

**Setting cronjob**

crontab –l

What we see now, is

no crontab for root. It is because there is no job defined.

crontab -e opens crontab editor. (at the first time, we need to set the default editor. I prefer vim :) )

Let's put there

12 9 23 \* \* /usr/bin/ls -al > logfile 2>&1

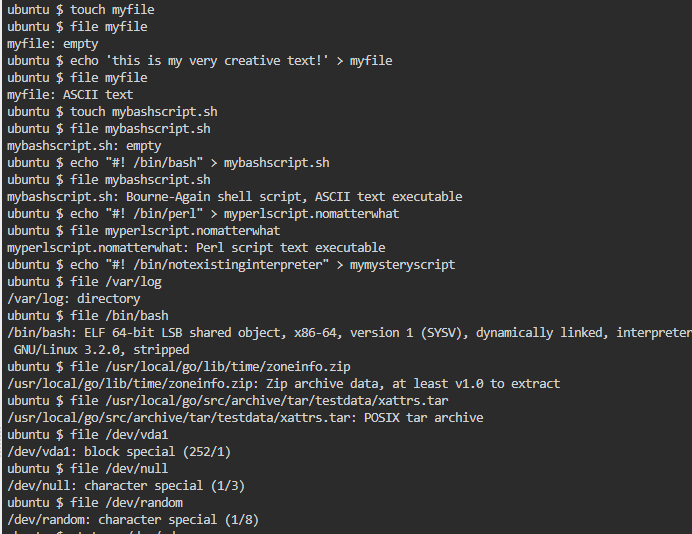
And save the file with :wq

You should see installing new crontab

Is it set?

.crontab -l

# Know your files!



touch myfile

And let's use our first command on the file

file myfile

In response we have myfile: empty. System knows only one thing - there is nothing inside.

Ok, we will put some content to the file.

echo 'this is my very creative text!' > myfile

What we will see after running file again?

file myfile

System correctly recognized the file as myfile: ASCII text.

Let's see what happen if we create script.

touch mybashscript.sh

file mybashscript.sh

So, we see the same thing. System doesn't care about file extension (lucky us, believe me), but about content.

Right. Now we do the script, shall we?

echo "#! /bin/bash" > mybashscript.sh

This is all we need :)

# Get information with stat

stat mybashscript.sh

Well, the output is much more rich than from file.

stat has parameters, like most of commands in Linux. Let's look on some.

stat -f mybashscript.sh

-f gives us information about filesystem.

stat -t mybashscript.sh

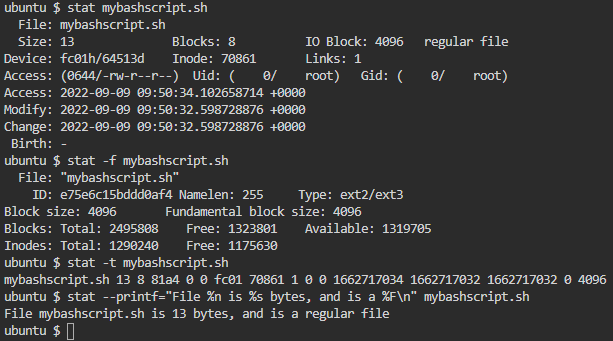
Will return condensed version of the information. Very cool to automation and use in scripts.

mybashscript.sh 13 8 81a4 0 0 fc01 70875 1 0 0 1652607024 1652607012 1652607012 0 4096

We can customize the output

stat --printf="File %n is %s bytes, and is a %F\n" mybashscript.sh

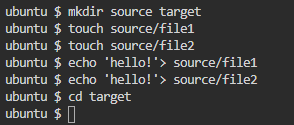
File mybashscript.sh is 13 bytes, and is a regular file.



# Soft and hard links

Let's start with some preparations. We will create two directories and files, which we will use to create links.

mkdir source target  
touch source/file1  
touch source/file2  
echo 'hello!'> source/file1  
echo 'hello!'> source/file2  
cd target



ln SOURCE TARGET

Here we have the syntax of the link command. Afterln command we have to provide the *source* (original) object and then the *target*. Target will be the actual link.

cat ../source/file1

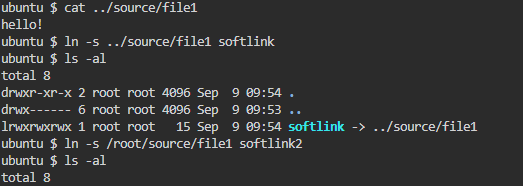
It is simple, but we don't need anything more.

Ok, we are ready to create the soft link.

ln -s ../source/file1 softlink

What we have now?

ls -al



ln -s /root/source/file1 softlink2

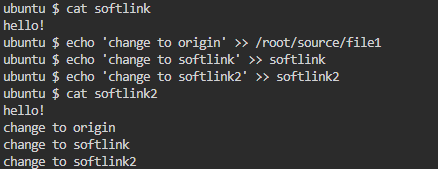
ls -al

cat softlink

And do some modifications

echo 'change to origin' >> /root/source/file1  
echo 'change to softlink' >> softlink  
echo 'change to softlink2' >> softlink2

cat softlink2



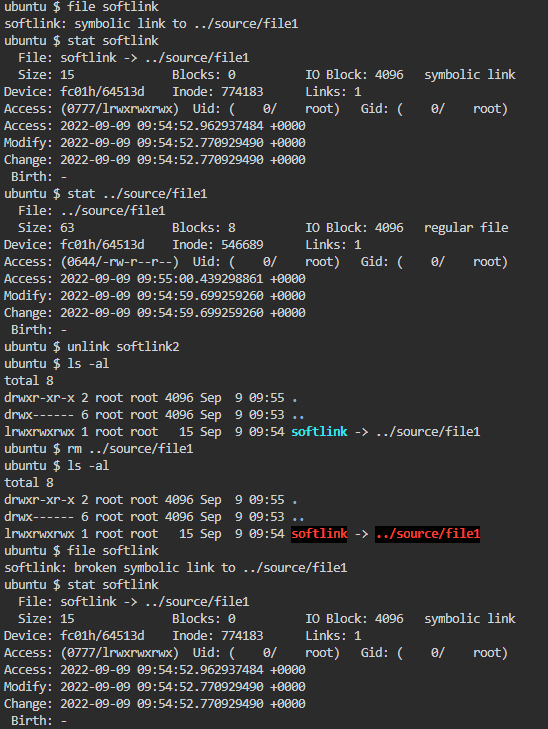
## Inodes

Although thee concept of inodes isn't familiar to us yet, but for now it is not needed. Let's see output of these two commands

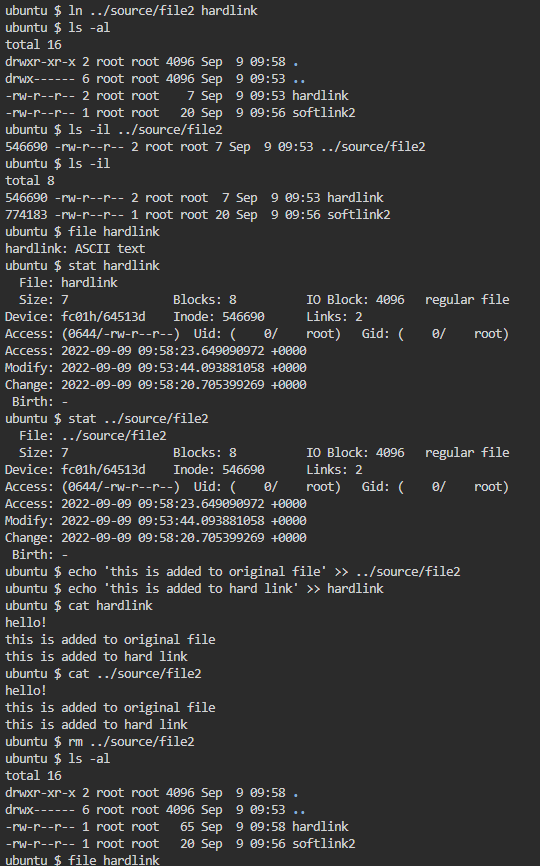
ls -il ../source/file1  
ls –il

file softlink

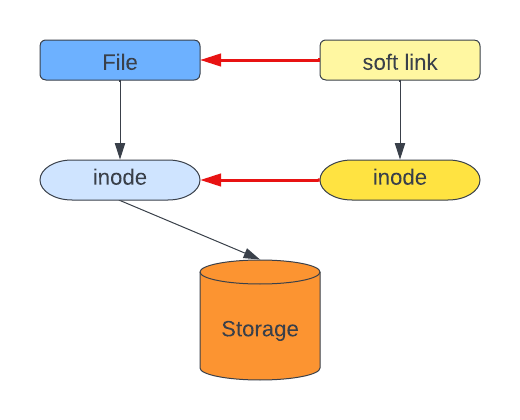
stat softlink



# Hard links

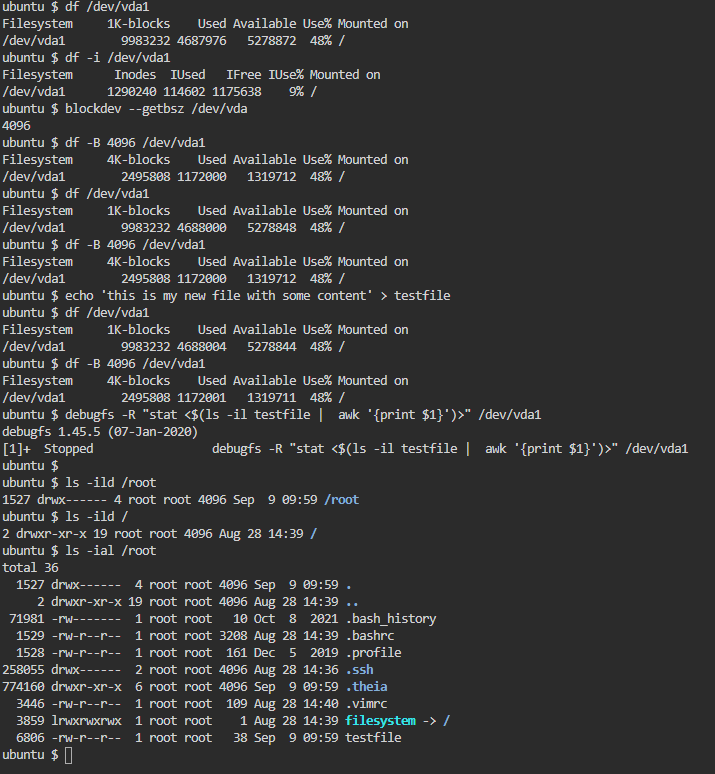


Ok, here is what happened. Let's think about inode as an element of communication between filesystem and physical storage. Take a look on the picture below. This is how soft link looks



We see that link on the file level is related to the original file. Both files have their own inodes and we can say (it is simplification) that inode of link is a shortcut to inode of original file. Only one inode points to the object on storage - original one.

# Inodes



df /dev/vda1

And now with inodes

df -i /dev/vda1

Let's do a small excercise.

df /dev/vda1  
df -B 4096 /dev/vda1  
echo 'this is my new file with some content' > testfile  
df /dev/vda1  
df -B 4096 /dev/vda1

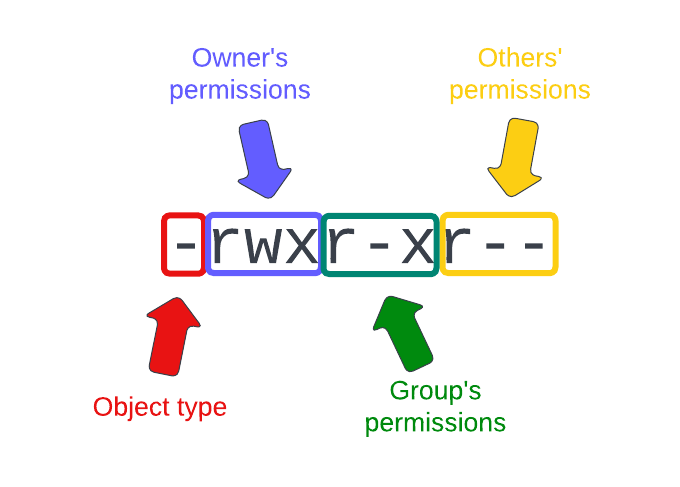
# Permissions

## Matrix explained

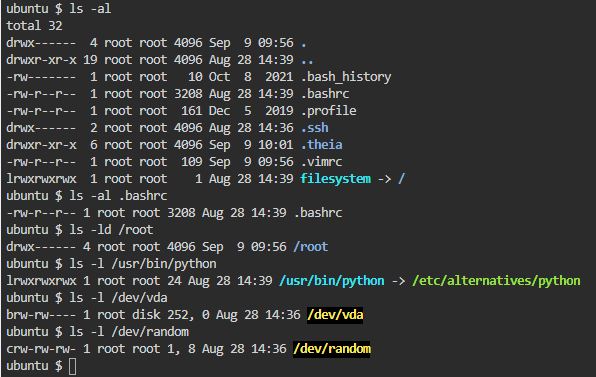
Let's take a look on the matrix itself. When we run

ls -al

We see the long format of the command ls. On the beginning of each line we see something similar to the picture below. Let's take a look.



I splitted this string of mysterious characters into four elements.



# Theory, continuation

Right. We've done a lot. Something still to do.

Let's list something again.

ls -l /var/log

Please, try to answer, how the system knows who is this owner? Or the group?

The answer is here:

-rw-r--r-- 1 root adm 48194 May 2 10:20 dmesg

Let's look on the picture.

