Laboratory Work №2.
Introduction to OS Linux.
Working with files in UNIX/Linux.

Goal: introduction to the UNIX command line, commands to work with the file System and commands for processing text files.

Introduction to bash

Shell

If you use Linux, then you know that after login you are greeted with an invitation

Command interpreter. For example:

```
\$
```

If graphical shell is loading after login, then to get to the command interpreter it needs to start the terminal emulator (gnome-terminal, xfce4-terminal,konsole, xterm, rxvt...).

The shell prompt on your computer may be different from the one shown in the example. It can contain the user name, computer name and the name of the current working directory.

But despite all these differences, the program which prints the invitation called "shell" (shell), and most likely in the role your shell performs a program called bash.

bash

You can check whether bash is running with the following command:

```
\$ echo \$SHELL /bin/bash
```

If as a result of this command you received an error or its output is different from that in the example, it is possible that your system does not use bash as the shell. Despite this, most of the material will be relevant, but still recommend that you switch to bash. You can do this (if bash is installed on the system) with the command:

```
\$ bash
```

What is bash?

Bash (an acronym for "**Bourne-again SHell**") is a standard command interpreter on most Linux systems. His duties include processing and executing commands with which the user controls the computer. After you have completed the work, you can end the shell process. After pressing the **Ctrl-D** key, **exit** or **logout,** the command interpreter process will be terminated and the user will be prompted to enter the user name and password.

Working with the Linux command line.

The description of the commands is divided into several sections - by the functional purpose of the commands.

File System Commands

pwd

The pwd (print working directory) command lets you find out the name of the current directory:

```
[user@localhost ~]$ pwd
/home/user
```

You are in the root directory now.

If in the travel process you lost and do not know where you are, just type this command, which will display the full name of the current directory to the screen, starting from the root directory.

If, instead of the name of the current directory, a diagnostic message appears on the screen:

Can not open .. and Read error in .., then there was a file system failure, maybe you do not have enough rights to read this directory.

cd

The cd command is used to change the current directory, i.e. To move to another directory and is an embedded shell command (an analog of the corresponding command for DOS-systems).

As an argument to this command, specify the name of the directory that you want to transfer to. For example:

cd /etc - move to the /etc directory, i.e. make it current.

If you want to move to a subdirectory, then "/" is not specified. Suppose you are in the /usr directory and you need to go to the local directory, which is a subdirectory of this directory. In this case, you can do the following:

```
[user@localhost usr]$ pwd
/usr
[user@localhost usr]$ cd local
[user@localhost local]$ pwd
/usr/local
```

Thus the /usr/local directory has become current.

If you enter the **cd** command without arguments, you will return to your home directory. This uses the system variable environment HOME.

```
[user@localhost usr]$ cd
[user@localhost ~]$ pwd
/home/user
```

Let's take a few more examples of using this command:

- **cd ..** — return to the parent directory (go up one directory up the tree);
- **cd ../..** — go up two catalogs up the tree;
- **cd /** — go to the root directory;
- **cd ../local** — go into the parent directory and move to its subdirectory local.

ls

The ls command provides various information about files and directories (similar to the **DIR** command for DOS-systems). Like most commands, ***ls*** has possible arguments and parameters (flags), which allow you to change its action.

Go to the root directory and see its contents using the command cd /. If no options are specified, the command displays the file and directory names that are sorted alphabetically.

```
[user@localhost ~]$ cd /
[user@localhost /]$ ls
bin  dev home  lib  mnt proc sbin sys usr
boot etc image media opt root srv  tmp var
```

Unfortunately, it is not yet clear what is the catalog, and what is the file. The next command can give information in the expanded format, which contains more information about each file (access rights, size, date of last modification, etc.):

```
[user@localhost /]$ ls -l
total 6
drwxr-xr-x  2 root root   40 Feb 10 17:57 bin
drwx------  2 root root  172 Feb 10  2007 boot
drwxr-xr-x  8 root root 3200 Feb 10 17:58 dev
drwxr-xr-x 50 root root  480 Feb 10 18:00 etc
drwxr-xr-x  3 root root   60 Feb 10 18:00 home
dr-xr-xr-x  5 root root 2048 Feb 10  2007 image
drwxr-xr-x 11 root root   40 Feb 10 17:57 lib
drwxr-xr-x  2 root root   40 Feb 10  2007 media
drwxr-xr-x  5 root root  100 Feb 10 17:57 mnt
drwxr-xr-x  2 root root    3 Feb 10  2007 opt
dr-xr-xr-x 53 root proc    0 Feb 10 17:56 proc
drwxr-xr-x  3 root root   40 Feb 10 17:57 root
drwxr-xr-x  2 root root   40 Feb 10 17:57 sbin
drwxr-xr-x  2 root root    3 Feb 10  2007 srv
drwxr-xr-x 11 root root    0 Feb 10 17:56 sys
drwxrwxrwt  7 root root  140 Feb 10 18:00 tmp
drwxr-xr-x 13 root root  152 Feb 10  2007 usr
drwxr-xr-x 19 root root   40 Feb 10 17:57 var
```

Columns from left to right: file type and access rights, number of hard links, owner name, owner group, byte size, modification time, name.

If you specify the file name as an operand, ***ls*** will only give information about this file:

```
[user@localhost /]$ ls -l bin/ls
-rwxr-xr-x 1 root root 73704 Feb 10  2007 bin/ls
```

If you specify the name of the directory as the operand, the contents of this directory will be issued, i.e. File names in this directory:

```
[user@localhost /]$ ls -l bin
total 4366
-rwxr-xr-x 1 root root   2852 Feb  1 23:01 arch
-rwxr-xr-x 1 root root  10732 Feb 18  2005 aumix-minimal
lrwxrwxrwx 1 root root      4 Feb 10  2007 awk -> gawk
-rwxr-xr-x 1 root root  13004 Feb 10  2007 basename
-rwxr-xr-x 1 root root 458000 Feb 14  2006 bash
...
```

If no operands are specified, the contents of the current directory will be displayed. If several operands are specified, information about the files is

displayed first, and then about the directories. The output is in alphabetical order.

You can use the following options:

-C

Output of the contents of the catalog in several columns. It is accepted by default when outputting to the terminal.

-F

Add a slash (/) to the directory name, an asterisk (*) to the name of the executable file, a sign (@) to each symbolic link.

-R

A recursive traversal of the subdirectories encountered. It can be useful for creating a list of all the system files.

-c

The output is sorted by the time of the last modification of the file.

-d

Output only the directory name (but not the content).

```
[user@localhost /]$ ls -ld bin
drwxr-xr-x 2 root root 40 Feb 10 17:57 bin
```

-f

The output is not sorted, i.e. The output order corresponds to the order of the files in the directory.

```
[user@localhost /]$ ls -f
.  bin  dev home  lib   mnt proc sbin sys usr
.. boot etc image media opt root srv  tmp var
```

-i

For each file, the index descriptor number (unique file number) is displayed.

```
[user@localhost /]$ ls -i
 27 bin     2 etc     11 lib    484 opt    31 sbin  1063 tmp
1216 boot   32 home   867 media    1 proc   485 srv   1215 usr
1467 dev   1856 image  829 mnt     33 root    1 sys    10 var
```

-r

Change the order of sorting to reverse (alphabetically or by time of last use).

-t

Sort by the time of the last modification of the file (the last modified is displayed first).

-1

Output one name on each page. It is accepted by default when output is not to the terminal.

mkdir

In the process of developing the system, you will need to create your own directory structure. After installing the system, there are quite a few catalogs in it and it's still difficult for them to start a new user. In the process of expanding the file system, you have to constantly create new ones and destroy old directories, so we'll take a look at how this can all be done.

It's quite easy to create a directory. For this, there is a command ***mkdir*** (its analog is also in DOS systems). As an argument, you must specify the name of the directory being created:

mkdir [-p] directory_name

```
[user@localhost /]$ cd
[user@localhost ~]$ mkdir test
[user@localhost ~]$ ls
Documents  test  tmp
```

The standard dot elements (.) for the directory itself and two dot (..) for its parent directory are created automatically. The **mkdir** command can also be used to create the underlying subdirectories directly from the current directory, specifying the full path to them. In this case, all directories specified in the path must exist and be accessible. If you need to create a hierarchy of directories, it's convenient to use the ***-p*** option:

```
[user@localhost ~]$ mkdir -p test/xxx/yyy/zzz
            [user@localhost ~]$ ls -R test
test:
xxx

test/xxx:
yyy

test/xxx/yyy:
zzz

test/xxx/yyy/zzz:
```

Without the -p option, a warning will be issued when attempting to create an already existing directory.

rmdir

After creating different catalogs, after a while you will have a natural desire to remove some of them. The command **rmdir** can be useful. Its format is as simple as the format of the previous **mkdir** command:

rmdir directory_name

The ***rmdir*** command deletes the directory whose name is specified as a parameter. The directory to be deleted must be empty, that's why, before destroying it, you must delete all files in it. In fact, this should be done very seldom, as the two of these problems is successfully controlled by the **rm** command, which will be reviewed later.

```
[user@localhost ~]$ rmdir test
rmdir: test: Directory not empty
```

Specified directory names are processed in order. At the same time, deleting the directory and its subdirectory, the subdirectory should be deleted earlier.

touch

The touch command only does what changes the time of the last access to the file. A remarkable feature of this command is that with its help you can create new empty files – if as an argument the name of a nonexistent file is transferred:

```
[user@localhost test]$ ls
xxx
[user@localhost test]$ touch file.test
[user@localhost test]$ ls -l
total 0
-rw-r--r-- 1 user user  0 Feb 10 18:26 file.test
drwxr-xr-x 3 user user 80 Feb 10 18:23 xxx
```

rm

In the process of working with the system quite often there is a need to delete files.

For this, the **rm** command exists, which allows you to delete both files and directories. It needs to be used with great caution, since UNIX-systems do not have a password, in contrast to Windows, to reverse the user before deleting the file, but do it quickly and for good. Therefore, the connection with the fact that on UNIX-systems of non-recoverable programs for the recovery of deleted files, think carefully before you delete something.

The rm command has the following format:

rm [-f | -i] [-dRr] file name ...

With this command, you can delete files whose names are specified as parameters. If the file is write-protected and the terminal is the standard input device, the user will be prompted to confirm the deletion of the file.

When you try to delete a directory using this command, you will receive an error message.

You can use the following options:

-d

If the file to be deleted is a directory, you can also delete it.

-f

Delete the write-protected file without confirmation. If this file does not exist, do not display a message about its absence. The *-f* option overrides the previous ***-i*** option.

-i

Confirm confirmation to delete any file, regardless of whether the terminal is a standard input device or not. The ***-i*** option overrides the previous ***-f*** option.

-r

Delete all files and subdirectories of this directory, and then delete the directory itself. The ***–R*** option implies the ***-d*** option. If the ***-i*** option is also specified, the user is prompted to confirm the deletion of files and directories.

cp

The program copies the contents of the file to a file with a different name, or to another directory with the preservation of the existing file name, and is also used to copy directories with their contents. The program has the following format:

cp [-ipr] file_name ... destination_file_name

You can use the following options:

-i

The user is asked to confirm when overwriting existing files and directories.

-p

Keep the existing mode of access to the file.

-r

Copying a directory with its subdirectories.

Copy the file to the current directory with a new name:

```
[user@localhost ~]$ cp /etc/resolv.conf resolv
```

Copy the file to the current directory with the name saved:

```
[user@localhost ~]$ cp /var/log/apache/access_log .
```

Copying the directory:

```
[user@localhost ~]$ cp -r test test.old
```

mv

Rename a file or move one or more files (or directories) to another directory. Command format:

mv [-i | -f] file_name ... destination_file_name

During migrating within a single file system, the command only changes the file pathname, so renaming and transfer are identical in implementation.

You can use the following options:

-f

Prohibit confirmation request when overwriting existing files.

-i

The user is asked to confirm when overwriting existing files and directories.

An example of using a command to transfer a file to a directory:

```
[user@localhost ~]$ mv text/user.html text/htmls/
```

ln

This command creates links to files both hard and symbolic. It has the following format (similar to the **mv** command in the order of the arguments):

ln [-s] file_name file_name_link

You can use the following options:

-f

Prohibit confirmation request when overwriting existing files (links).

-s

Creation of a symbolic link.

Examples of using the command:

```
[user@localhost ~]$ ln text/alex/linux.html working/linux-todo.html
```

Creating a symbolic link to the directory:

```
[user@localhost ~]$ ln -s images/my/photos photos
```

du

The command displays the size of the space on the disk occupied by the directory (and all its subdirectories) in blocks (by default, 1 block is 512 bytes). By default, information about the current directory is displayed.

The program has the following parameters:

-a

Output information not only about directories, but also about files.

-h

Output information about catalog size in the familiar format: kilobytes, megabytes, etc.

-s

Output only the grand total, without temporary information.

Example of program execution:

```
[user@localhost ~]$ du -sh test
925K    test
```

tar

The tar command is intended for archiving group files and directories. This command allows you to merge a group of files and directories with all the attributes into a single file that has the extension ".tar". The resulting file is then used with the same tar command. The tar command has many options, but we will focus on the following format for using the command:

tar [-c | -x] [-z | -j] -f tar_file_name file_and_directory_names

The ***-c*** option corresponds to the creation of the archive, ***-x*** - the deployment of the archive. The created archive can also be compressed using the gzip or bzip2 archivers. For this, the options ***-z*** and ***-j*** are used, respectively. Here we can clearly see the difference between the concept of "archiving", which is performed by the command tar, and "compression", which produces specialized programs. The name of the archive that is created or expanded is passed after the ***-f*** option. Let's consider some examples of working with this program:

Creation of the archive etc.tar.bz2 of the / etc directory using bzip2 compression:

```
[user@localhost ~]$ tar -cjf etc.tar.bz2 /etc
[user@localhost ~]$ ls
etc.tar.bz2
```

Expanding the archive etc .tar.bz2 into the current directory using bzip2 despreading:

```
[user@localhost ~]$ tar -xjf etc.tar.bz2
[user@localhost ~]$ ls
etc etc.tar.bz2
```

find

When working with files, it is often necessary to find something. On UNIX, there is a **find** command for this. In general, this command has the following format:

find [search] [search_condition] [action]

The first argument to the **find** command is the *name of the directory* in which the search is performed. By default, this is the current directory.

The following are the *search conditions* that can take the following values:

-name

File name. You can use templates in it, but you must enclose the name in double quotes.

-type

File type: **f** - regular file, **d** - directory, **l** - symbolic link, etc.

-user

The name of the owner of the file or its unique identifier (UID).

-group

Name of the group of the owner of the file or its unique identifier (GID).

-perm

File permissions.

-size

File size. Usually the number is followed by a letter - in which the size is measured (in blocks, bytes, kilobytes, etc.).

-atime

Time of access to the file.

-ctime

The time when the owner of the file last changed.

-mtime

The time when the contents of the file were last modified.

-newer

Search for all files newer than specified.

Search terms can be combined with modifiers: ***-a*** — AND, ***-o*** — OR, ***\!*** — NOT, ***\(...\)*** — group of conditions.

The following actions can be applied to the found files:

-print

Display the name of the found file along with the path. This action is performed by default.

-delete

Delete the found files.

-exec command {} \;

Execute the specified command for each found file with the transfer of the file name as an argument ({}).

-ok command {} \;

Similar to the -exec action, only for each file is the confirmation requested before executing the command.

Let's consider some examples of using the command:

Output of all files in the current directory and subdirectories:

```
[user@localhost ~]$ find
.
./xxx
./xxx/yyy
./xxx/yyy/zzz
./file.test
```

Output of all files in the /etc directory starting with "re":

```
[user@localhost ~]$ find /etc -name "re*"
find: /etc/tcb: Permission denied
find: /etc/default: Permission denied
/etc/chroot.d/resolv.all
/etc/chroot.d/resolv.conf
/etc/chroot.d/resolv.lib
/etc/redhat-release
/etc/remounttab
/etc/resolv.conf
...
```

Output of all subdirectories in the /etc directory starting with "re":

```
[user@localhost test]$ find /etc -name "rp*" -type d
/etc/rpm
```

Output of all files in the director y /etc, changed in the last day:

```
[user@localhost test]$ find /etc -mtime -1
/etc
/etc/issue
/etc/issue.net
/etc/mtab
```

Output of all files in the /tmp di rectory that are not owned by the user:

```
[user@localhost test]$ find /tmp \! -user user
/tmp
```

```
/tmp/.private
/tmp/.font-unix
/tmp/.X11-unix
```

Delete all obsolete files in the current directory (ending with "~"):

```
[user@localhost test]$ find -name "*~" -delete
```

Displaying the names of all text files in the home directory:

```
[user@localhost test]$ find ~ -name "*.txt" -exec echo {} \;
/home/user/a.txt
/home/user/unix_commands.txt
```

Copy all text files to a floppy disk:

```
[user@localhost test]$ find ~ -name "*.txt" -exec cp {} /mnt/floppy/ \;
```

Scenario: Introduction to UNIX directories

This scenario will highlight acquaintance with UNIX commands, studying the file system and base directories, creating a working environment in the user's home directory for all subsequent commands.

Initial conditions: Command line after logging in.

Get the name of the current directory using the **pwd** command:

```
[user@localhost ~]$ pwd
/home/user/
```

Go to the root directory with the **cd /** command:

```
[user@localhost ~]$ cd /
[user@localhost /]$ pwd
/
```

Mark how the prompt line changed.

View the contents of the root directory with the **ls** command:

```
[user@localhost /]$ ls
bin  dev home  lib   mnt proc sbin sys usr
boot etc image media opt root srv  tmp var
```

Compare using the "extended" output of the **ls -F** command:

```
[user@localhost /]$ ls -F
bin/  dev/ home/  lib/   mnt/ proc/ sbin/ sys/ usr/
boot/ etc/ image/ media/ opt/ root/ srv/  tmp/ var/
```

The directories are marked in blue and the "/" sign after the name.

View the contents of the home directory with the **ls ~** command:

```
[user@localhost /]$ ls ~
Documents  tmp
```

Documents tmp

The home directory contains a set of standard directories. Return to the home directory with cd without parameters:

```
[user@localhost /]$ cd
[user@localhost ~]$
```

Create a test dire ctor y with the **mkdir test** c ommand:

```
[user@localhost ~]$ mkdir test
```

View the updated content of the home directory **ls**:

```
[user@localhost ~]$ ls
Documents  test  tmp
```

Create a subtest subd irectory in the test dire ctory with t he **mkdir test /subte st** command:

```
[user@localhost ~]$ mkdir test/subtest
```

View the contents of the home directory and its subdirectories by using the recursive *–R* key in the ls -R command:

```
[user@localhost ~]$ ls -R
.:
Documents  test  tmp

./Documents:

./test:
subtest

./test/subtest:

./tmp:
```

Scenario: Exploring file types on UNIX

In this scenario, file types on UNIX are considered: simple files, directories, links. The commands for creating and copying files are being studied. The difference between hard and symbolic links, between copying and transferring a file is shown.

Initial conditions: Command line. The test directory after the previous script. Create an empty file using the touch **f**irst.t**x**t command:

```
[user@localhost test]$ touch first.txt
[user@localhost test]$ ls
```

```
first.txt  subtest
```

Add a line of text to the end of the file with the command echo "**H**ello, world" **>> f**irst.t**x**t and redirect the output:

```
[user@localhost test]$ echo "Hello, world" >> first.txt
```

View the contents of the file with the command **cat first. txt:**

```
[user@localhost test]$ cat first.txt
Hello, world
[user@localhost test]$
```

View advanced information about the directory using the *-l* option of the **ls** command. **ls -l test:**

```
[user@localhost test]$ ls -l
total 4
-rw-r--r-- 1 user user 13 Feb 14 20:12 first.txt
drwxr-xr-x 2 user user 60 Feb 14 19:42 subtest
```

Add the alias of the shell to reduce the size of the command, using the command alias **ls='ls -F -l'**:

```
[user@localhost test]$ alias ls='ls -F -l'
[user@localhost test]$ ls test
total 4
-rw-r--r-- 1 user user 13 Feb 14 20:12 first.txt
drwxr-xr-x 2 user user 60 Feb 14 19:42 subtest
```

Look deeper into the test directory using the keys *-a* and *-i*. **ls -a -i**

```
[user@localhost test]$ ls -a -i
total 4
1014 drwxr-xr-x 3 user user 100 Feb 14 20:07 ./
941 drwx------ 8 user user 340 Feb 14 19:28 ../
1081 -rw-r--r-- 1 user user  13 Feb 14 20:12 first.txt
1015 drwxr-xr-x 2 user user  60 Feb 14 19:42 subtest/
```

The first column is the index numbers of the file system. The third column is the number of hard links of the file.

Copy the file using the command **cp first.txt copy1.txt**:

```
[user@localhost test]$ cp first.txt copy1.txt
[user@localhost test]$ ls -a -i
total 8
1014 drwxr-xr-x 3 user user 120 Feb 14 20:33 ./
941 drwx------ 8 user user 340 Feb 14 19:28 ../
1082 -rw-r--r-- 1 user user  13 Feb 14 20:33 copy1.txt
1081 -rw-r--r-- 1 user user  13 Feb 14 20:12 first.txt
1015 drwxr-xr-x 2 user user  60 Feb 14 19:42 subtest/
```

The new file has its own index node.

Rename a file using the command mv first.txt orig.txt:

```
[user@localhost test]$ mv first.txt orig.txt
[user@localhost test]$ ls -a -i
total 8
1014 drwxr-xr-x 3 user user 120 Feb 14 20:37 ./
941 drwx------ 8 user user 340 Feb 14 19:28 ../
1082 -rw-r--r-- 1 user user  13 Feb 14 20:33 copy1.txt
1081 -rw-r--r-- 1 user user  13 Feb 14 20:12 orig.txt
1015 drwxr-xr-x 2 user user  60 Feb 14 19:42 subtest/
```

Note that only the name of the file has changed, all other attributes remain the same.

Create a hard link with the command **ln orig.txt copy2.txt**:

```
[user@localhost test]$ ln orig.txt copy2.txt
[user@localhost test]$ ls -a -i
total 12
1014 drwxr-xr-x 3 user user 140 Feb 14 20:41 ./
941 drwx------ 8 user user 340 Feb 14 19:28 ../
1082 -rw-r--r-- 1 user user  13 Feb 14 20:33 copy1.txt
1081 -rw-r--r-- 2 user user  13 Feb 14 20:12 copy2.txt
1081 -rw-r--r-- 2 user user  13 Feb 14 20:12 orig.txt
1015 drwxr-xr-x 2 user user  60 Feb 14 19:42 subtest/
```

One more link to the same file was added, the number of links increased by 1.

Create a symbolic link with the command **ln -s orig.txt orig.lnk:**

```
[user@localhost test]$ ln -s orig.txt orig.lnk
[user@localhost test]$ ls -a -i
total 12
1014 drwxr-xr-x 3 user user 160 Feb 14 20:45 ./
941 drwx------ 8 user user 340 Feb 14 19:28 ../
1082 -rw-r--r-- 1 user user  13 Feb 14 20:33 copy1.txt
1081 -rw-r--r-- 2 user user  13 Feb 14 20:12 copy2.txt
1083 lrwxrwxrwx 1 user user   8 Feb 14 20:45 orig.lnk -> orig.txt
1081 -rw-r--r-- 2 user user  13 Feb 14 20:12 orig.txt
1015 drwxr-xr-x 2 user user  60 Feb 14 19:42 subtest/
```

The new file has a new index node and a size equal to the name of the file orig .txt.

Compare the contents of files by accessing them by name:

```
[user@localhost test]$ cat orig.txt
Hello, world.
[user@localhost test]$ cat copy2.txt
Hello, world.
[user@localhost test]$ cat orig.lnk
Hello, world.
```

The same data can be accessed through a hard or symbolic link.

Scenario: Searching for system logs

In this scenario, the file and directory search command is examined.

Initial conditions: Command line, test directory after the last script. List all files and directories in the current directory, including the contents of the subdirectories with the **find** command:

```
[user@localhost test]$ find
.
./orig.lnk
./copy2.txt
./orig.txt
./copy1.txt
./subtest
```

Find all the files and directories in the current directory and its subdirectories that start with "o" using the command **find -name "o*"**:

```
[user@localhost test]$ find -name "o*"
./orig.lnk
./orig.txt
```

Find all the files and directories in the / etc directory and its subdirectories that begin with "o" using the command **find /etc -name "o*"**:

```
[user@localhost test]$ find /etc -name "o*"
find: /etc/tcb: Permission denied
find: /etc/default: Permission denied
find: /etc/buildreqs: Permission denied
...
```

The list of found files can be too large and you can use the Shift-PgUp and Shift-PgDn keys to scroll the terminal. You have seen a lot of "Permission denied" error messages that can be suppressed by redirecting errors as follows: **find /etc - name "o*" 2>/dev/null**

```
[user@localhost test]$ find /etc -name "o*" 2>/dev/null
/etc/modprobe.d/options
/etc/modutils.d/oss
/etc/net/ifaces/default/fw/options
/etc/net/ifaces/default/options
...
/etc/pam.d/other
/etc/rc.d/init.d/outformat
```

Find all the directories in /etc that start with "o" using the command **find /etc –name "o*" -a -type d 2>/dev/null**:

```
[user@localhost test]$ find /etc -name "o*" -a -type d 2>/dev/null
/etc/net/options.d
```

```
/etc/openssh
/etc/openssl
/etc/opt
```

Find all the regular files in the /var directory and its subdirectories ending with "log":

```
[user@localhost test]$ find /var -name "*log" -a -type f 2>/dev/null
/var/log/Xorg.0.log
/var/log/faillog
/var/log/lastlog
```

Create the logs directory with the command **mkdir logs**:

```
[user@localhost test]$ mkdir logs
```

Copy the found files to the local directory by using the -exec option of the find command. To do this:

find /var -name "*log" -a -type f -exec cp {} test/logs/2>/dev/null

```
[user@localhost test]$ find /var -name "*log" -a -type f -exec cp {} test/logs/ 2>/dev/null
[user@localhost test]$ ls test/logs
итого 789
-rw-r-----  1 user users 601033 Oct 16 18:37 emerge.log
-rw-r--r--  1 user users 292292 Oct 16 18:37 lastlog
-rw-r--r--  1 user users  37383 Oct 16 18:37 Xorg.0.log
```

We copied all the files that we have the rights to read.

Scenario: Archiving and de-archiving files and directories

In this scenario, the command for archiving files and directories is being studied. The command line, the test directory after the last script.

Initial conditions: Go to the home directory with the command cd:

```
[user@localhost test]$ cd
[user@localhost ~]$
```

Create an archive named test.tar.gz using compression, containing the test directory with the command **tar -czf test.tar.gz test**:

```
[user@localhost ~]$ tar -czf test.tar.gz test
[user@localhost ~]$ ls
drwxr-xr-x  4 user users   208 Окт 16 18:36 test/
-rw-r--r--  1 user users 79173 Окт 16 18:49 test.tar.gz
```

Create a new directory for the contents of the archive with the command **mkdir test2**:

```
user@desktop ~ $ mkdir test2
```

Go to the new directory using the command **cd test2**:

```
user@desktop ~ $ cd test2
```

Expand the contents of the archive to the current directory using the command **tar -xzf ../test.tar.gz**:

```
user@desktop test2 $ tar -xzf ../test.tar.gz
```

Make sure that the contents of the unpacked archive coincide with the original directory. Enter the command **ls -l -F**:

```
user@desktop test2 $ ls -l -F
итого 0
drwxr-xr-x  4 user users 208 Окт 16 18:36 test/
user@desktop test2 $ ls -l -F test
итого 12
-rw-r--r--  1 user users  13 Окт 15 20:54 copy1.txt
-rw-r--r--  2 user users  13 Окт 15 20:48 copy2.txt
drwxr-xr-x  2 user users 136 Окт 16 18:37 logs/
lrwxrwxrwx  1 user users   8 Окт 16 18:54 orig.lnk -> orig.txt
-rw-r--r--  2 user users  13 Окт 15 20:48 orig.txt
drwxr-xr-x  2 user users  48 Окт 13 21:33 subtest/
```

Scenario: Creating new text files

In this scenario, commands are examined for creating and modifying text files. Command line.

Initial conditions: Enter the command echo "One line":

```
user@desktop test $ echo "One line"
One line
```

This command takes a string as an argument and prints it to standard output. Enter the same command, but redirect the output to the second.txt file with ">". Enter the command **echo "One line" > second.txt**:

```
user@desktop test $ echo "One line" > second.txt
```

Add the line "A line" to the end of file second.txt with another redirect echo "A line" >> second.txt:

```
user@desktop test $ echo "A line" >> second.txt
```

Output the contents of a file using the command cat second.txt:

```
user@desktop test $ cat second.txt
One line
A line
```

Using the cat command, you can create multi-line files - if you redefine the program output to a file and enter text before pressing Ctrl-D (end of input). Introduce the team **cat >multiline.txt** and type the text

```
user@desktop test $ cat >multiline.txt
Simple text:
blah-blah-blah
1 2 3 4 5 6 7 8 9 0

bye!

(Ctrl+D)
user@desktop test $
```

Ensure that the contents of the file are the same as the text entered, including all line feeds. To do this, enter the command **cat multiline.txt**

```
user@desktop test $ cat multiline.txt
Simple text:
blah-blah-blah
1 2 3 4 5 6 7 8 9 0

bye!
```

1. The main purpose of the cat command is to combine files whose names are passed as command-line arguments. Merge files with the command **cat orig.txt second.txt multiline.txt > big.txt**:

```
user@desktop test $ cat orig.txt second.txt multiline.txt > big.txt
```

2. Make sure that the new file contains the lines from the listed files using the command **cat big.txt**:

```
user@desktop test $ cat big.txt
Hello, world
One line
A line
Simple text:
blah-blah-blah
1 2 3 4 5 6 7 8 9 0

bye!
```

QUIZ

1. Create the **test1** directory in the home directory. Compare the time creation of the system directories /bin, /tmp with time creation of the directory test1.

2. Copy the /bin /ls file to the local directory. Look at the attributes of this file. Try to start it.

3. Create a symbolic link **tmplnk** in the local directory to /tmp. Copy several files to the **tmplnk** directory.

4. Compare the files /dev/tty1 and /dev/hda1. What type do they have? What is the difference?

5. Find all the files on the system that have been modified no more than a day ago.

6. Using one command, find all the files with the .html extension in the /usr directory and copy them to the local htmls directory.

7. Create an archive of the /etc directory and find out its size. Try creating a compressed archive of the same directory. Compare the compression ratio of gzip and bzip2 to this example.