“Київський фаховий коледж зв’язку”

Циклова комісія Комп’ютерної та програмної інженерії

ЗВІТ ПО ВИКОНАННЮ

ЛАБОРАТОРНОЇ РОБОТИ №10

з дисципліни: «Операційні системи»

Тема: “Зміна власників і прав доступу до файлів в Linux. Спеціальні каталоги та файли в Linux”

Виконали

студенти

групи РПЗ-03

Команда 6: Sichkar Maxim,

Brytyuk Bohdan

Kyiv 2023

**The goal of the work:**

1. Getting practical skills for working with the Bash command shell.

2. Getting to know the basic actions when changing file owners.

3. Getting to know the basic actions when changing access rights to files

4. Familiarity with special directories and files in Linux.

**Material provision of classes**

1. IBM PC type computer.

2. OS family Windows (Windows 7).

3. Virtual machine - Virtual Box (Oracle).

4. GNU/Linux operating system - CentOS.

5. Cisco network academy site netacad.com and its online Linux courses

**Tasks for preliminary preparation.**

1. Read the short theoretical information for the laboratory work and make a small dictionary

basic English terms for command assignments and their parameters.

2. Study the materials of the online course "NDG Linux Essentials" (netacad.com):

- Chapter 17 - Ownership and Permissions

- Chapter 18 - Special Directories and Files

3. Take the test in the NDG Linux Essentials course on the following topics:

- Chapter 17 Exam

- Chapter 18 Exam

4. On the basis of the considered material, answer the following questions:

**4.1. What is the purpose of the id command?**

The id command is a Unix/Linux utility that is used to display the user and group IDs (UID and GID) associated with the current user or with a specified username. It is a simple and useful command-line tool that helps users determine their own permissions and group membership, or the permissions and group membership of other users on the system.

Here are some common uses of the id command:

Display UID and GID information: By running the id command without any arguments, you can view the current user's UID and GID information.

Display username from UID: You can use the id -un <UID> command to display the username associated with a specific UID.

Display group membership: You can use the id -G command to display a list of GIDs for the current user, which indicates the groups the user is a member of.

Display supplementary group membership: You can use the id -Gn command to display a list of supplementary groups for the current user, which indicates the groups that the user is not the primary member of, but still has access to.

In summary, the id command is used to display user and group ID information in Unix/Linux systems. It can display information about the current user or about a specified username, and can show user and group membership information as well.

**4.2. How to see what access rights the owner of a file has?**

To see the access rights that the owner of a file has in a Unix/Linux system, you can use the ls command with the -l (long format) option. This will display detailed information about the file, including the access permissions for the owner, group, and other users.

Here's an example of how to use ls to see the access rights for the owner of a file:

$ ls -l myfile.txt

-rw-r--r-- 1 owneruser ownergroup 1024 May 3 10:30 myfile.txt

In this example, the access permissions for owneruser (the file owner) are -rw-, which indicates that the owner has read and write permissions on the file, but does not have execute permissions. The r-- and r-- following the owner permissions indicate the read-only permissions for the group and other users, respectively.

In summary, you can use the ls -l command to see the access rights that the owner of a file has, as well as the access rights for other users and groups on the system.

**4.3. How to change the group owner?**

To change the group owner of a file or directory in a Unix/Linux system, you can use the chgrp command followed by the name of the new group you want to assign.

Here's the syntax for using the chgrp command:

chgrp [OPTIONS] GROUP FILE

GROUP: the name or ID of the new group that you want to assign as the group owner

FILE: the name of the file or directory that you want to change the group ownership for

Here's an example of how to use chgrp to change the group owner of a file named myfile.txt to a group named newgroup:

$ chgrp newgroup myfile.txt

In this example, the chgrp command changes the group owner of the file myfile.txt to newgroup. Note that to run this command, you must have the appropriate permissions to change the group ownership of the file.

In summary, to change the group owner of a file or directory, you can use the chgrp command followed by the new group name or ID and the name of the file or directory that you want to change the group ownership for.

**4.4. How can you view the current file type in the terminal? Give examples for different types files**

To view the current file type in the terminal in a Unix/Linux system, you can use the file command followed by the name of the file you want to check.

Here's the syntax for using the file command:

file [OPTIONS] FILENAME

FILENAME: the name of the file that you want to check

Here are some examples of different types of files and how you can check their type using the file command:

$ file myfile.txt

myfile.txt: ASCII text

In this example, the file command shows that myfile.txt is a text file with ASCII encoding.

$ file myfile.bin

myfile.bin: data

In this example, the file command shows that myfile.bin is a binary file, but does not provide any further information about its content.

$ file myimage.jpg

myimage.jpg: JPEG image data, JFIF standard 1.01

In this example, the file command shows that myimage.jpg is a JPEG image file.

$ file myarchive.tar.gz

myarchive.tar.gz: gzip compressed data, from Unix, original size 10240

In this example, the file command shows that myarchive.tar.gz is a gzip-compressed file.

In summary, you can use the file command in Unix/Linux systems to view the current file type of a file in the terminal. The output of the file command will provide information about the type of file and its content, which can help you determine how to work with the file.

**4.5. What are Setuid and Setgid permissions used for?**

Setuid and setgid permissions are used to grant users or processes additional permissions beyond their normal level of access.

Setuid (set user ID) is a special permission bit that can be set on an executable file. When a program with setuid permission is executed, it runs with the effective user ID of the owner of the file rather than the user who launched it. This allows ordinary users to execute programs or scripts with elevated privileges. For example, the passwd command has the setuid bit set so that a regular user can change their own password without requiring root privileges.

Setgid (set group ID) is a special permission bit that can be set on a directory. When a file or subdirectory is created within a directory with setgid permission, the new file or directory is assigned the same group owner as the parent directory. This allows multiple users to work together in a shared directory with a common group ownership. For example, a project directory can be owned by a project group with setgid permission, so that any files or subdirectories created within it automatically inherit the same group ownership.

However, it is important to note that setuid and setgid permissions can be a security risk if not used properly. Programs or scripts with setuid permission should be carefully reviewed and tested to ensure that they do not have any security vulnerabilities. Similarly, setgid directories should be used with caution to ensure that users only have access to the files and directories that they need.

**4.6. Why does the system need the so-called "sticky bit" (Sticky Bit). Give examples of when this it is advisable to use permission.**

The "sticky bit" is a special permission bit that can be set on a directory. When the sticky bit is set on a directory, it means that only the owner of a file can delete or rename that file within that directory, even if other users have write permission on the directory. This helps prevent accidental deletion or modification of files by other users.

One example of when it may be advisable to use the sticky bit is on directories used for temporary files, such as the /tmp directory. The sticky bit can be set on the /tmp directory to prevent users from accidentally deleting or modifying files that belong to other users.

Another example is on public directories, such as those used for file sharing or collaborative work. In this case, the sticky bit can be set to ensure that only the owner of a file can modify or delete it, while still allowing other users to read or copy the file.

To set the sticky bit on a directory, you can use the chmod command with the +t option, followed by the name of the directory. For example:

chmod +t /path/to/directory

To remove the sticky bit, you can use the chmod command with the -t option, followed by the name of the directory. For example:

chmod -t /path/to/directory

In summary, the sticky bit is used to ensure that only the owner of a file can delete or rename that file within a directory, even if other users have write permission on the directory. It can be useful in situations where multiple users are sharing a directory or when a directory is used for temporary files.

5. Prepare the initial version of the report electronically:

- Title page, topic and purpose of the work

- Glossary of terms

- Answers to clauses 4.1-4.6 from tasks for preliminary training

Progress.

1. Initial work in CLI mode in Linux OS of the Linux family:

1.1. Start the VirtualBox virtual machine, select CentOS and run it. Log in

under user: CentOS, password for login: reverse (if you run LR in 401 aud.) and run

terminal.

1.2. Start the Ubuntu\_PC virtual machine (if you are doing the LR tasks through the netacad academy)

1.3. Start your Linux family operating system (if you are working on your own PC and its

installed) and launch the terminal.

2. Work through all the command examples presented in the NDG Linux course labs

Essentials:

- Lab 17: Ownership and Permissions

- Lab 18: Special Directories and Filesteams\*\*\*

The name of the command Its purpose and functionality

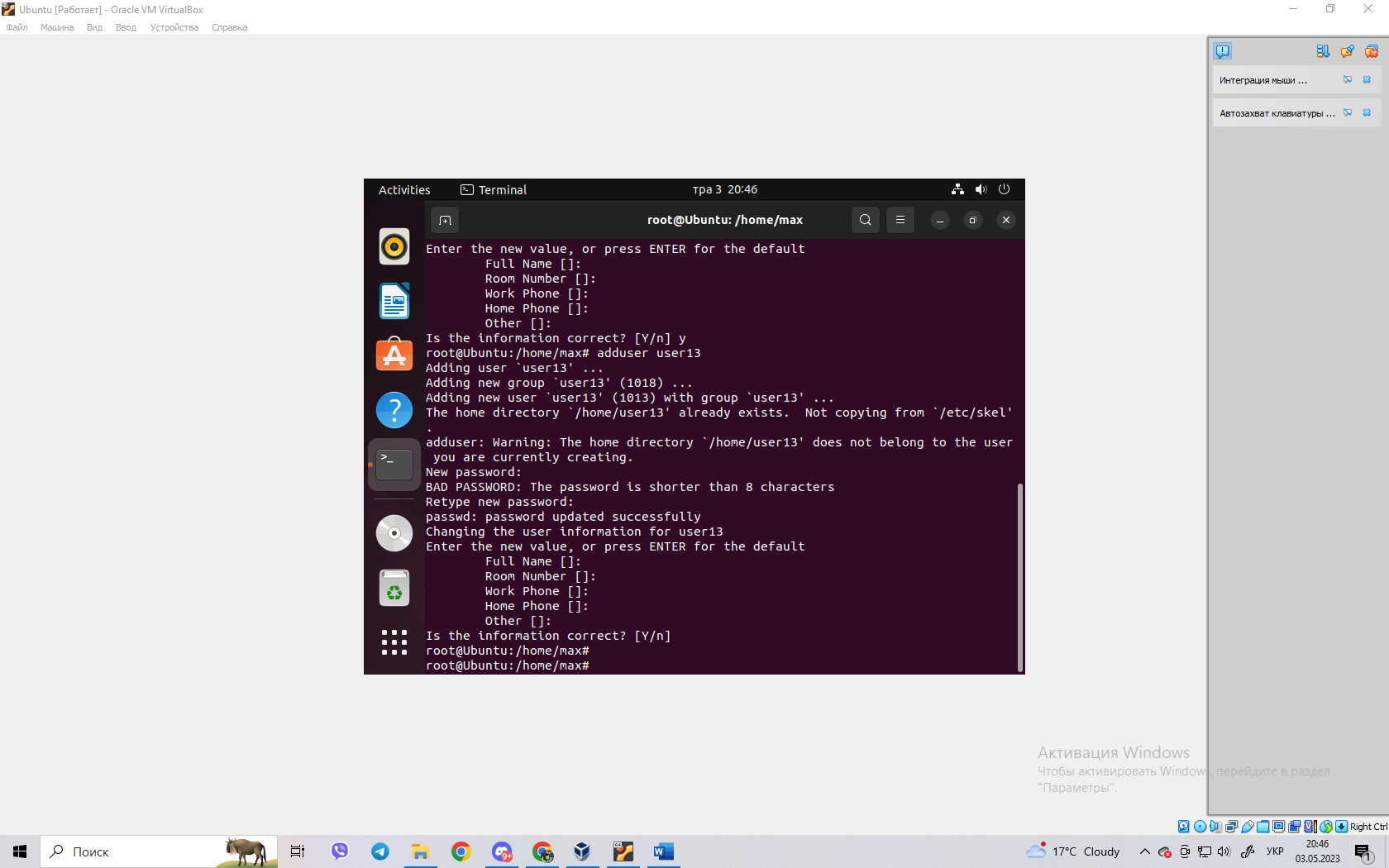
|  |  |
| --- | --- |
| The name of the command | Its purpose and functionality |
| chmod | The chmod command in Linux is used to change the access permissions of a file or directory. It stands for "change mode", which is a way to specify and modify the permissions granted to users, groups, and others to access a file or directory. |
| echo "date" > test.sh | The command "chmod echo "date" > test.sh" will not work as intended.  The "chmod" command is used to change the permissions of a file or directory, but the command given is attempting to execute a command and redirect its output to a file named "test.sh".  To create a file named "test.sh" with the content "date" and set its permissions, you can use the following commands:  echo "date" > test.sh  chmod 755 test.sh  The first command creates a file named "test.sh" with the content "date". The second command sets the permissions of the file to read, write, and execute for the owner of the file, and read and execute for everyone else. |
| test.sh | test.sh is likely a shell script file that contains a series of commands and instructions to be executed by the shell interpreter. The purpose and functionality of the test.sh script depend on its specific contents, which could be any set of commands or functions written in a shell scripting language such as Bash, sh, or zsh.  Typically, shell scripts are used to automate tasks, perform system administration tasks, or execute a series of commands in a specific sequence. The test.sh script may contain commands to manipulate files, start or stop services, configure system settings, or perform any other task that can be executed via a command-line interface.  Without knowing the specific contents of the test.sh script, it is impossible to provide a more detailed explanation of its purpose and functionality. |

\*\*\*Screenshots of the execution of commands in the terminal may not be presented, it is enough to briefly describe the commands

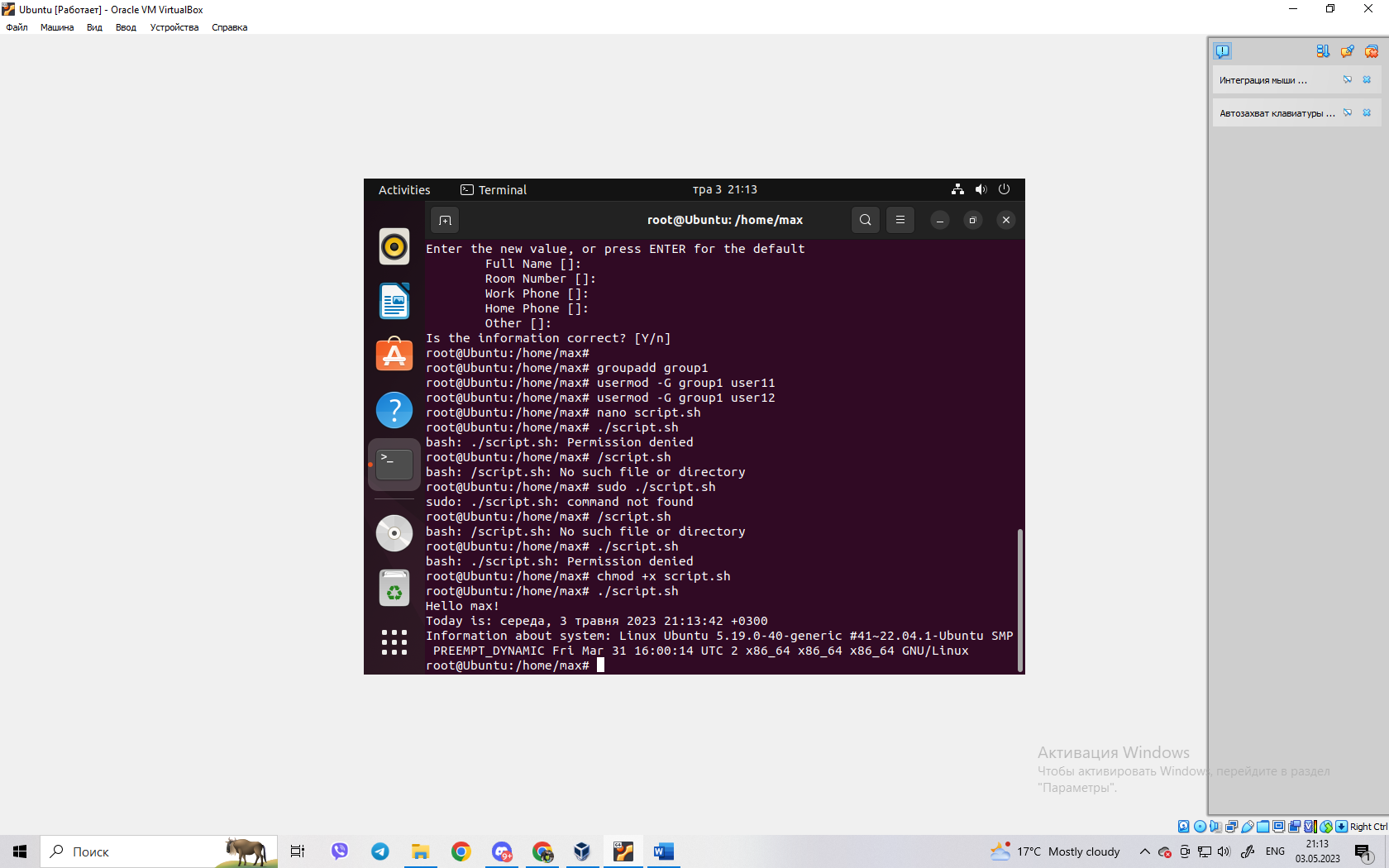
in the table.

**4. Perform the following practical tasks in the terminal following actions (show screenshots):**

- create three new users;

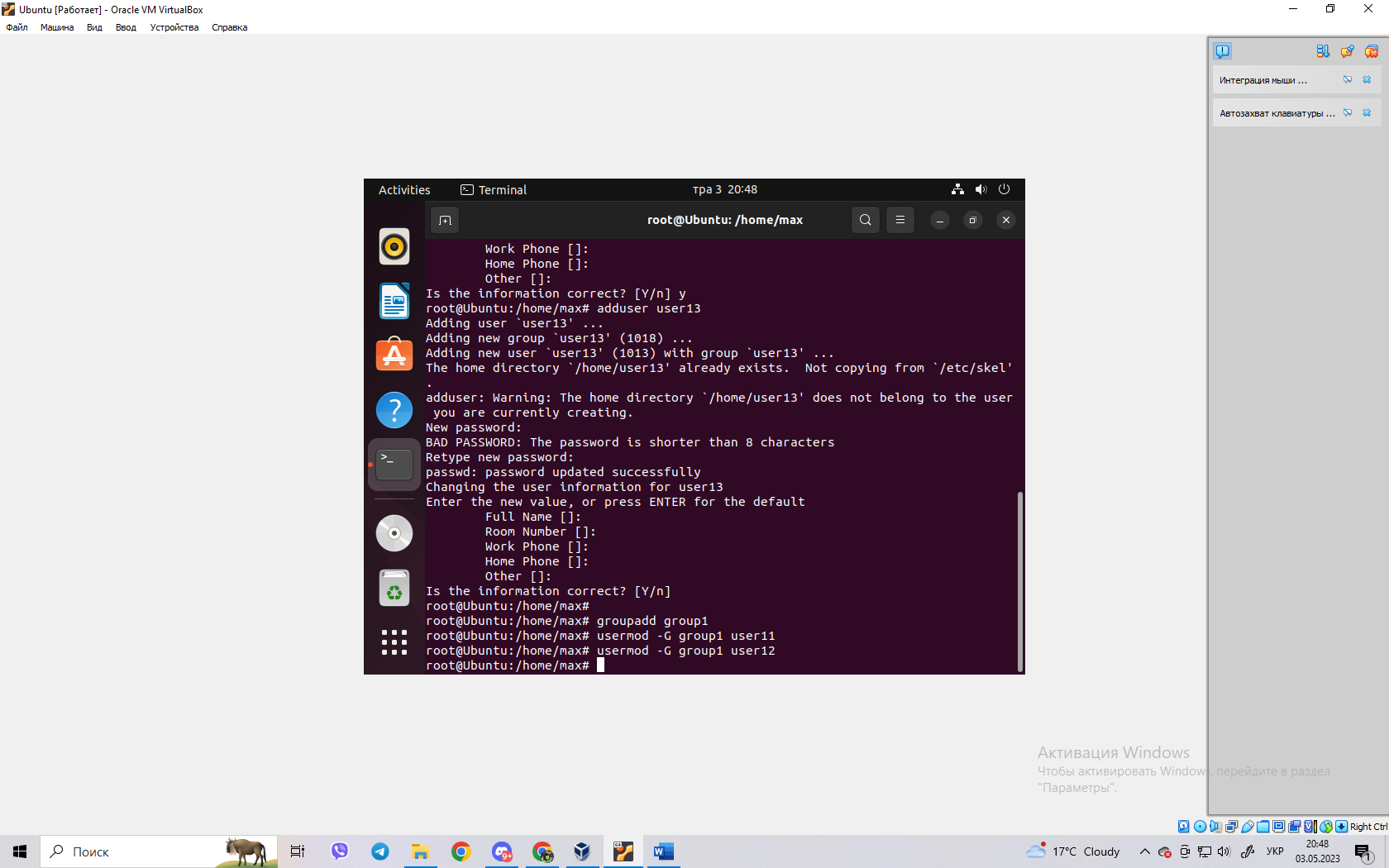
****

- create a new user group, add two of the three created users there;



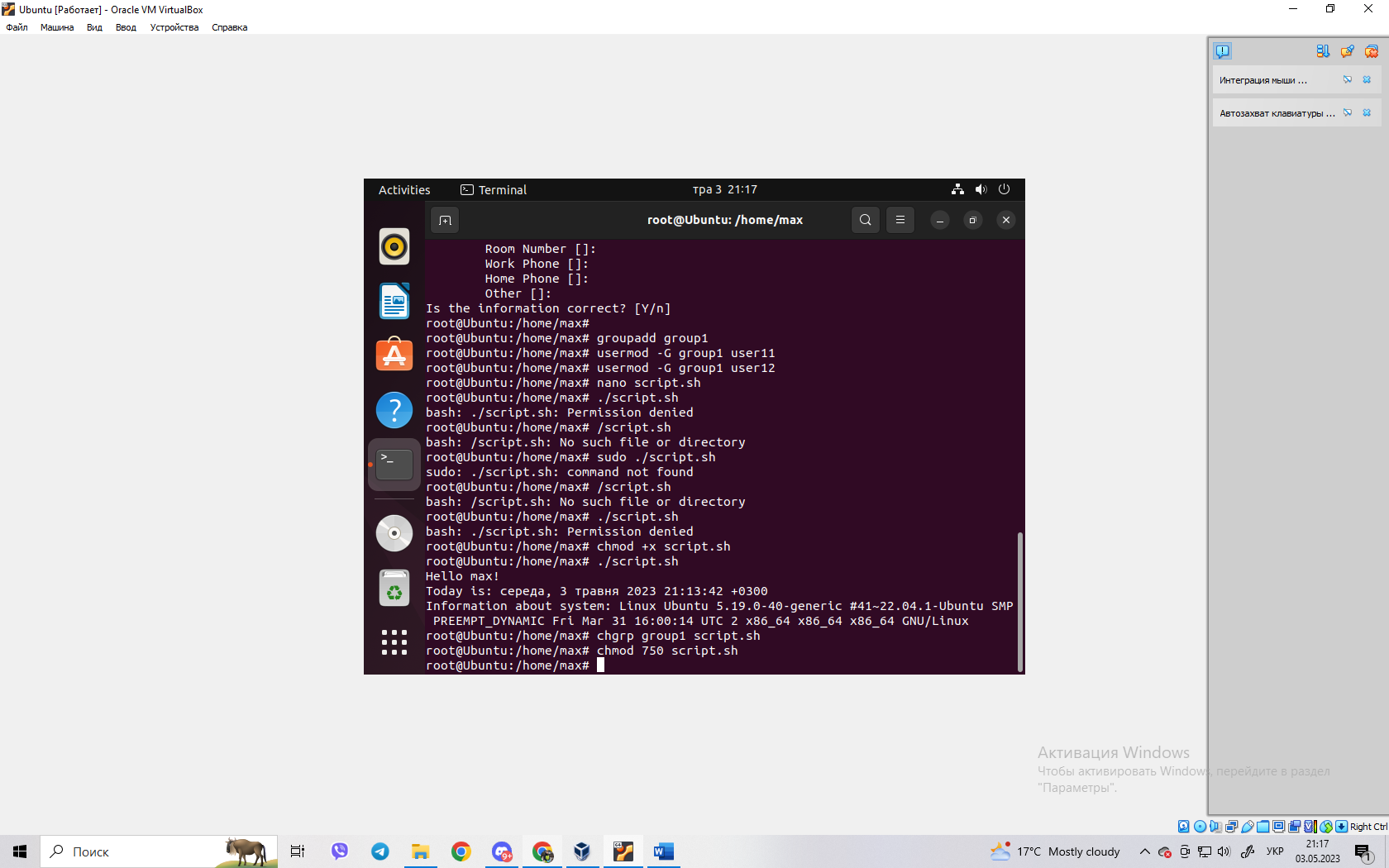
- create a new file that will be available for reading, editing and execution by the owner

file, for example, the simplest script script;

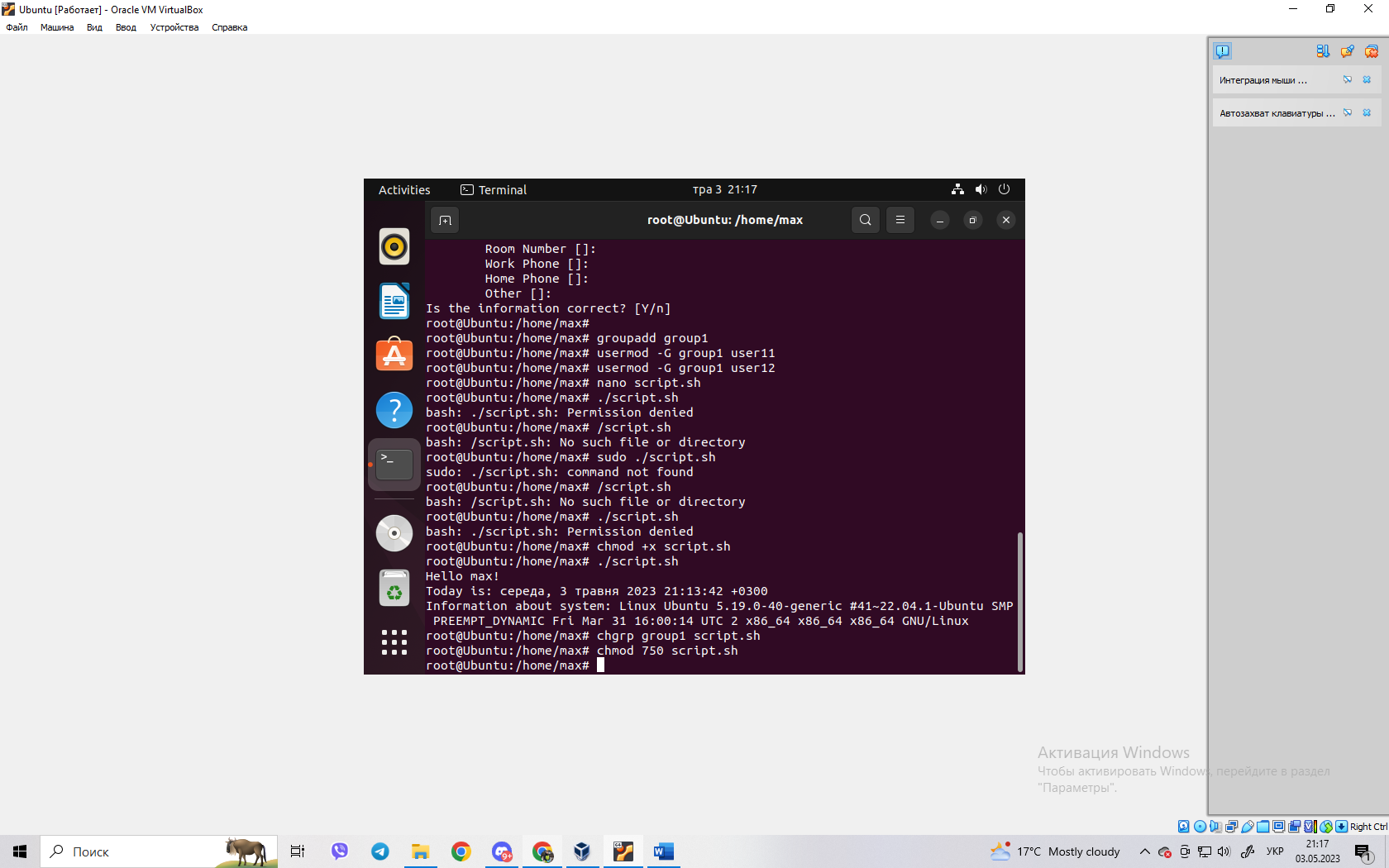


- for users in the owner group, grant view and execute permissions (without permission to

editing) of this file;



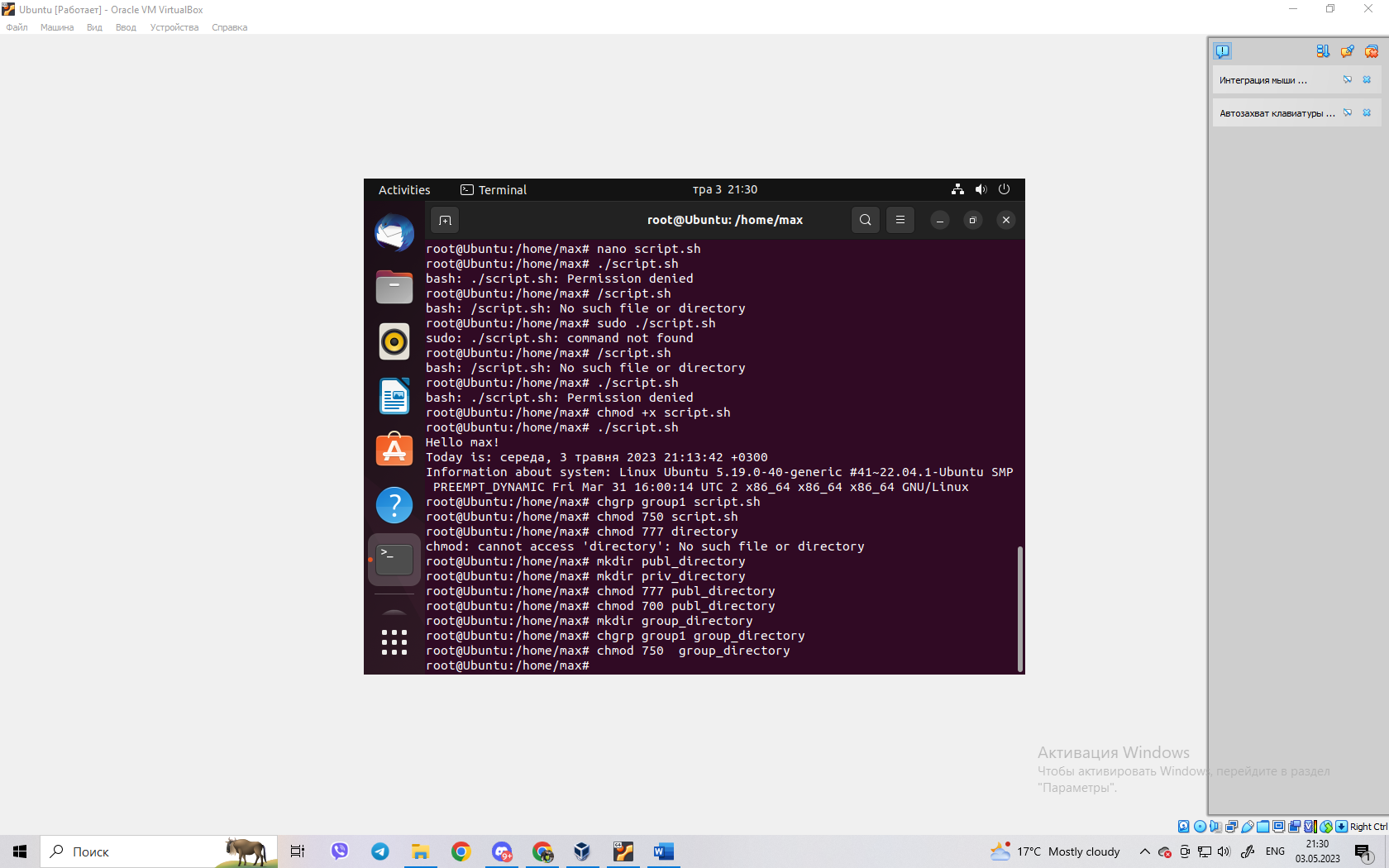
- prohibit access to this file for other users;



- perform similar actions for directories - create a directory that will be accessible to all three

users, create a directory that will be accessible only to the owner, create a directory that

users of the owner group will be able to view, but not edit;



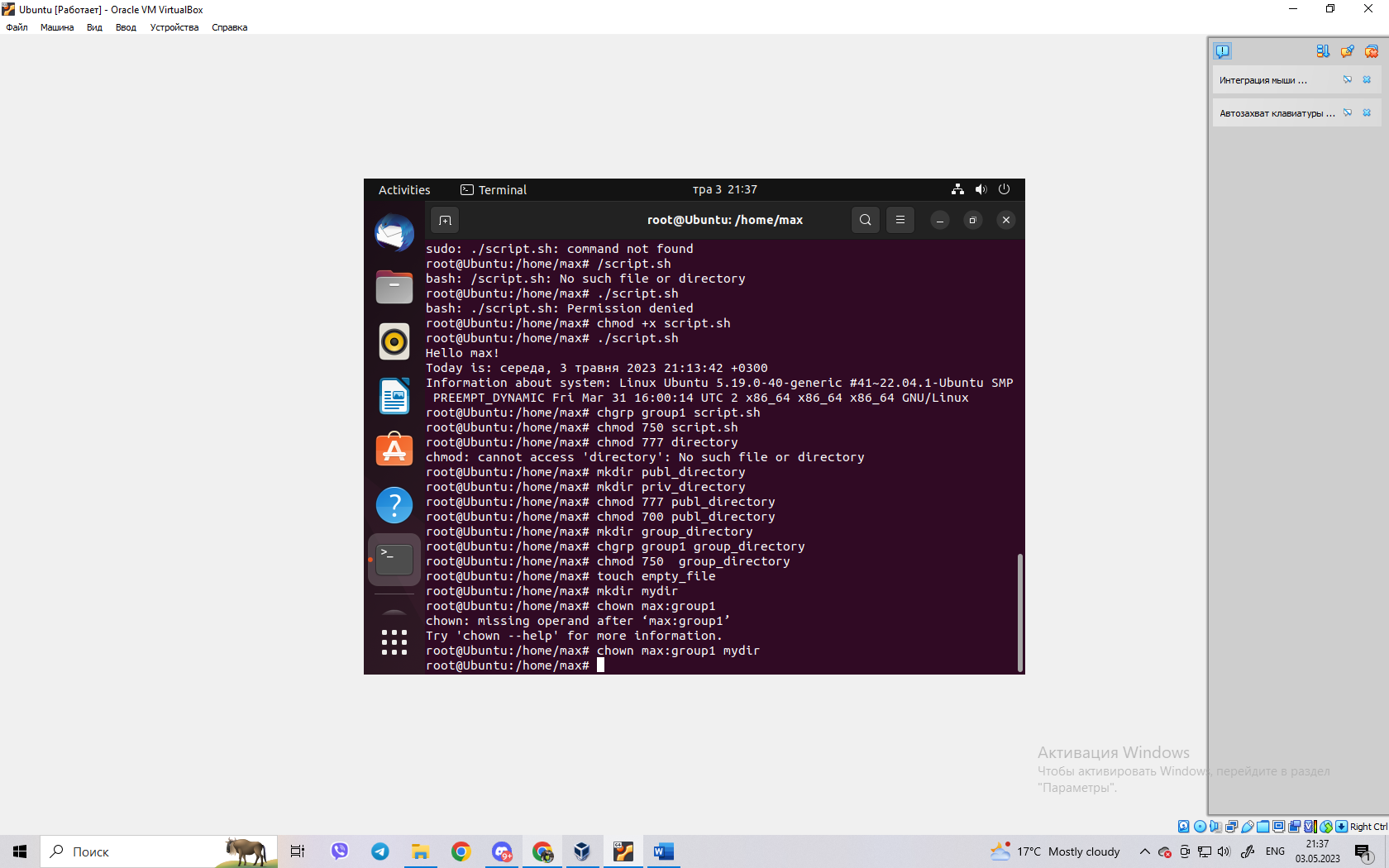
- create an empty file called emptyfile using the touch emptyfile command. Now

"reset" file permissions with chmod 000 emptyfile. What happens if you change permissions for emptyfile by passing only one value for chmod in numeric mode, for example chmod 4

emptyfile? What if we use two numbers, for example chmod 44 emptyfile? That we

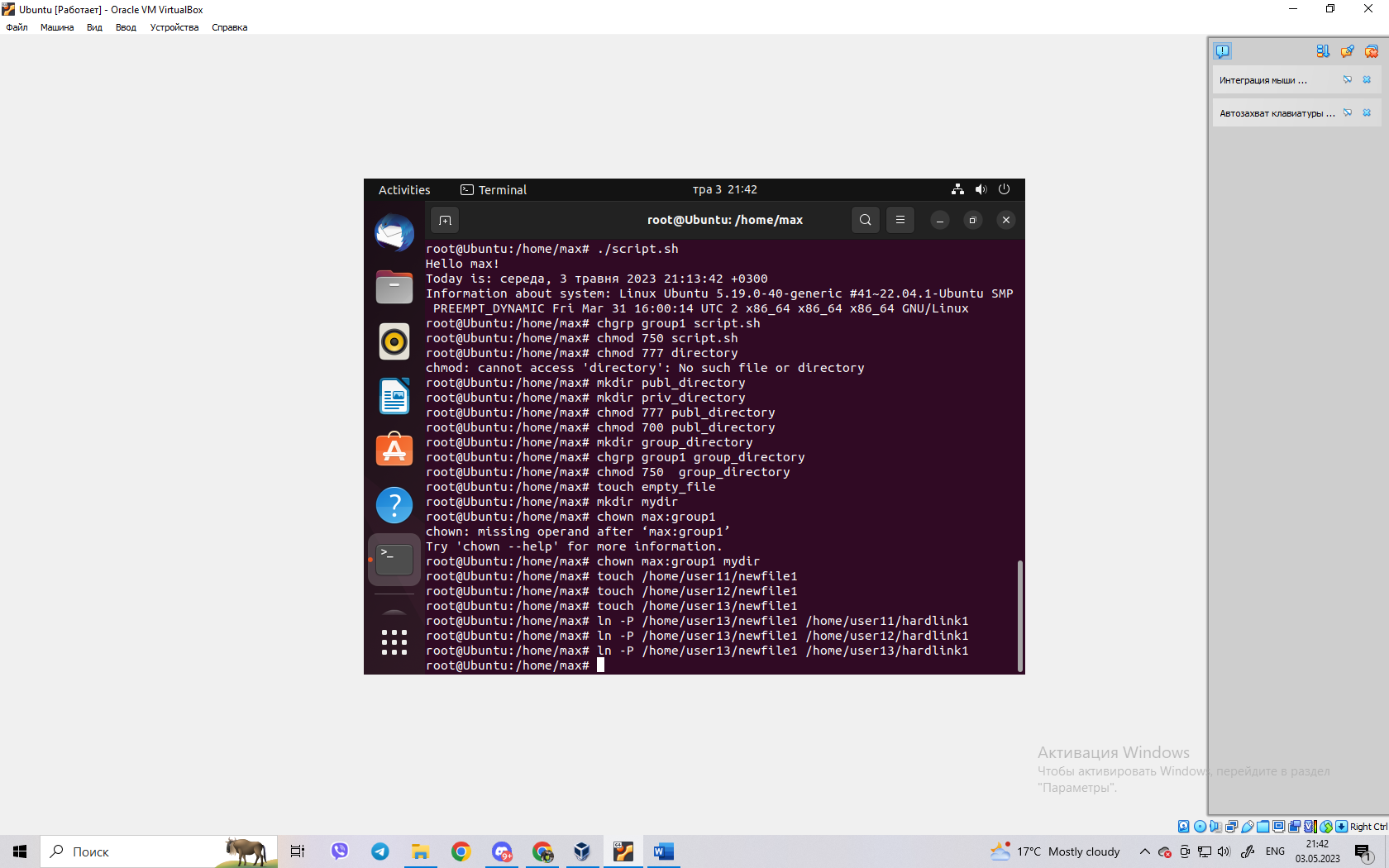
can we learn how chmod reads a numeric value?

After executing the chmod 000 emptyfile command, all permissions on the file will be disabled, meaning that no user will be able to read, edit, or execute the file. If we use a single number in the chmod command in numeric mode, it means that only the file owner's permissions will be changed. In our case, if we execute the command chmod 4 emptyfile, then the read permission will be given only to the owner of the file, and all other permissions will remain disabled. If we use two numbers in the chmod command in numeric mode, the first number indicates the permissions for the owner of the file, and the second number indicates the permissions for users belonging to the file owner's group and all other users. In our case, if we execute the chmod 44 emptyfile command, read permissions will be granted to both the file owner and users belonging to the file owner's group, and all other permissions will remain disabled.

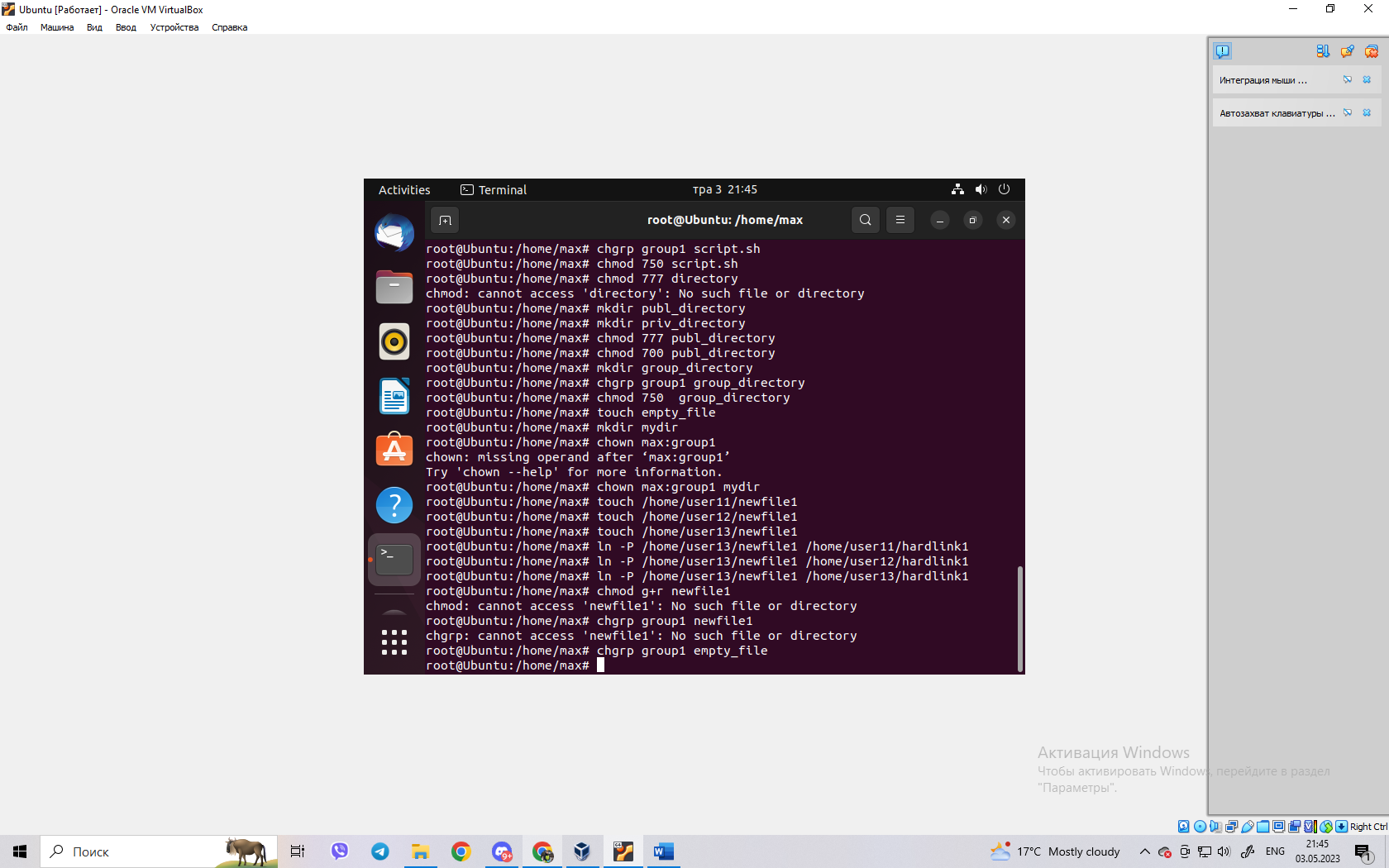


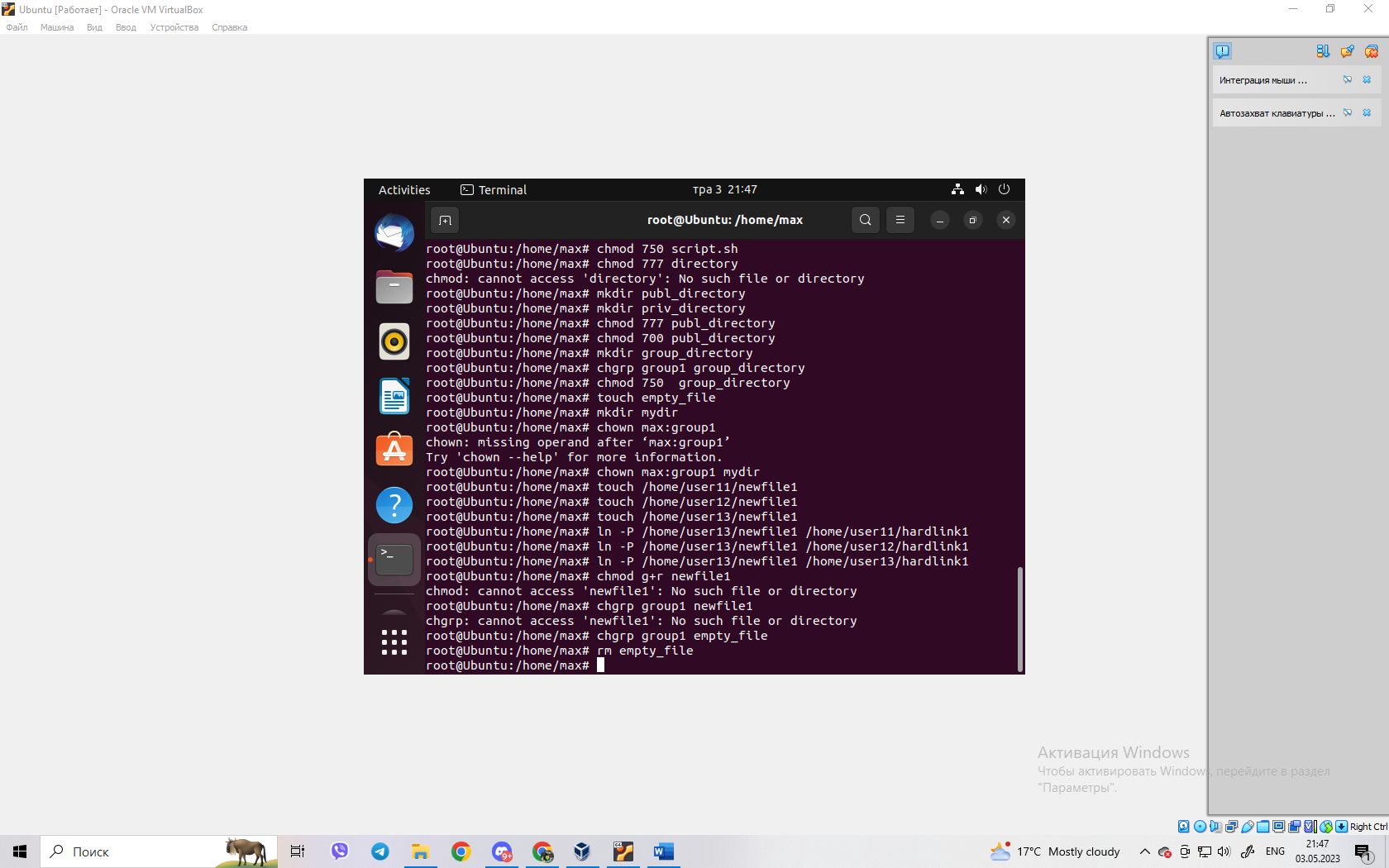
- under each user, create one new file, and a hard and symbolic link

on him;



- try other users to view these files;



- try other users to delete these files, 

1. **Give examples of changing access rights by symbolic method (Symbolic Method)?**

The symbolic method of changing access rights uses a combination of letters and symbols to modify the permissions of a file or directory. The letters correspond to the user or group that you want to modify the permissions for, while the symbols indicate the type of modification you want to make. Here are some examples of how to use the symbolic method to change access rights:

To add read permission for the owner of a file:

chmod u+r filename

The u stands for "user" and the +r adds read permission.

To remove write permission for the group that a file belongs to:

chmod g-w filename

The g stands for "group" and the -w removes write permission.

To add execute permission for all users:

chmod a+x filename

The a stands for "all" and the +x adds execute permission.

To set read, write, and execute permissions for the owner, and only read and execute permissions for the group and others:

chmod u=rwx,g=rx,o=rx filename

The = sets the permissions to the specified value, while the , separates the different permissions for different users or groups. Here, u=rwx sets the permissions for the owner to read, write, and execute, g=rx sets the permissions for the group to read and execute, and o=rx sets the permissions for others to read and execute.

These are just a few examples of how to use the symbolic method to change access rights. It can be a powerful tool for managing file and directory permissions, but it's important to understand the syntax and how to use it correctly.

1. **Give examples of changing access rights by numerical method (numeric method, octal method)?**

The numerical or octal method of changing access rights is based on using a three-digit number to specify the permissions for the owner, group, and others. Each digit represents a different set of permissions: the first digit specifies the permissions for the owner, the second digit specifies the permissions for the group, and the third digit specifies the permissions for others. Each digit can be a value between 0 and 7, with each value corresponding to a different combination of permissions:

0: no permissions

1: execute

2: write

3: write and execute

4: read

5: read and execute

6: read and write

7: read, write, and execute

Here are some examples of how to use the numerical method to change access rights:

To give read, write, and execute permissions to the owner, and read and execute permissions to the group and others:

chmod 755 filename

Here, the first digit (7) sets the permissions for the owner to read, write, and execute, while the second and third digits (5) set the permissions for the group and others to read and execute.

To give read and write permissions to the owner, and no permissions to the group and others:

chmod 600 filename

Here, the first digit (6) sets the permissions for the owner to read and write, while the second and third digits (0) set the permissions for the group and others to no permissions.

To give execute permission to all users, but no other permissions:

chmod 111 filename

Here, all three digits (1) set the permissions for all users to execute.

These are just a few examples of how to use the numerical method to change access rights. It can be a more concise way of specifying permissions, but it's important to understand how the numbers correspond to different sets of permissions.

**3. Is it possible to execute a file that has execute rights but not set read rights (--x)?**

**Explain.**

Yes, it is possible to execute a file that has execute rights but not read rights (--x), provided that you know the exact filename and location of the file.

When you execute a file, the shell needs to read the contents of the file in order to load it into memory and execute it. However, if the file has no read permissions, the shell cannot read the contents of the file, and therefore cannot display any information about the file, such as its contents or permissions.

However, if you know the exact filename and location of the file, you can still execute it by specifying its full path when you run the file. For example, if the file is located at /home/user/file and has execute permissions but no read permissions, you can execute it by running:

/home/user/file

This will execute the file, but you won't be able to see its contents or permissions unless you have sufficient permissions to view them separately. It's worth noting that executing a file without reading its contents can be risky, as you may not know what the file does or what its effects may be.

1. **What is the purpose of the umask command?**

The umask command in Unix-like operating systems is used to set the default file permission mode for newly created files and directories. It stands for "User Mask" and is used to mask off or remove certain permission bits from the default permissions that are applied to new files and directories.

The umask value is a three-digit octal number that specifies the permissions to be removed from the default permission mode. The first digit represents the permissions to be removed from the owner of the file, the second digit represents the permissions to be removed from the group, and the third digit represents the permissions to be removed from others.

For example, a umask value of 022 means that the write permission will be removed from the group and others, and a umask value of 077 means that the read, write, and execute permissions will be removed from the group and others, and only the owner of the file will have any permissions.

The umask command is usually set in the shell initialization files such as .bashrc or .profile and it is executed automatically when a new shell is started. The umask command is useful in ensuring that new files and directories are created with the appropriate default permissions.

1. **If we change access rights and permissions in the current session, will they be saved in the next session?**

No, changing access rights and permissions in the current session will not persist across sessions unless they are saved explicitly.

When you change the access rights and permissions of a file or directory in a terminal session, these changes apply only to that specific session. If you close the terminal or log out of the system, the changes will be lost.

To make the changes persist across sessions, you need to use a command that explicitly sets the access rights and permissions on the file or directory, such as chmod or chown. These commands allow you to set the access rights and permissions in a way that will be saved and applied even after you log out or restart the system.

Additionally, you can also modify the default permissions for new files and directories using the umask command. This command sets a default mask that is applied to new files and directories, and can be configured to ensure that specific access rights and permissions are always set by default.

1. **Is there any pattern that the system uses regarding rights and accesses when creating new files. As is it possible to change the default permissions?**

Yes, there is a pattern that the system uses regarding rights and accesses when creating new files, and this pattern is determined by the default umask value.

The default umask value is a three-digit octal number that specifies which permissions are not allowed by default. When a new file is created, the system applies the default umask value to determine which permissions should be removed from the maximum permission set.

For example, a default umask value of 022 means that write permission is removed from the group and others, so new files will be created with the permissions 644 (rw-r--r--), where the owner has read and write permission, while the group and others have only read permission.

You can change the default permissions by setting a different umask value. For example, if you want all new files to be created with read and write permission for the owner and read-only permission for group and others, you can set the umask value to 022. On the other hand, if you want new files to be created with read and write permission for both owner and group, and read-only permission for others, you can set the umask value to 002.

You can set the default umask value for your user in your shell initialization files such as .bashrc or .bash\_profile, or system-wide in the /etc/profile file. To set the umask value, you can use the umask command followed by the desired octal value.

1. **Imagine that a program needs to create a one-time temporary file that is never created again will be needed after closing the program. What is the correct directory to create this file?**

For a one-time temporary file that is never created again after closing the program, it is best to use the /tmp directory. This directory is designed for storing temporary files and is typically cleared out periodically by the system to free up space.

You can use the mktemp command to create a unique temporary file in the /tmp directory. The mktemp command will create a file with a unique name, and you can then use this file for temporary storage during the execution of your program. Once your program has finished executing, you can delete the temporary file using the rm command.

For example, the following command creates a unique temporary file in the /tmp directory:

$ tmpfile=$(mktemp /tmp/myprogram.XXXXXX)

This will create a file with a name like /tmp/myprogram.Ou8NlG. You can then use the $tmpfile variable to read and write to the temporary file within your program. Once your program has finished executing, you can delete the temporary file using the following command:

$ rm "$tmpfile"

Note that the mktemp command is designed to create a temporary file with secure permissions, so you don't need to worry about setting the correct file permissions yourself.

1. **How can you create a hard link? In what situations is it advisable to use them?**

To create a hard link, you can use the ln command followed by the name of the file you want to link to, and the name of the new hard link. Here is an example:

$ ln /path/to/file /path/to/link

This creates a new hard link called link that points to the original file located at /path/to/file. The hard link and the original file share the same inode on the file system, so they essentially point to the same file. If you modify the contents of one file, the other file will be updated as well since they are actually the same file.

It is advisable to use hard links when you want to create another reference to an existing file without duplicating the file's contents. This can be useful when you want to have multiple names for the same file, or when you want to create a backup of a file without using additional disk space.

Note that hard links cannot be created across different file systems or partitions, and that removing the original file does not remove the hard link. If you delete the original file, the hard link will still exist and will still point to the file's data until it is also deleted.

1. **How can you create a symbolic link? In what situations is it advisable to use them?**

To create a symbolic link, you can use the ln command followed by the name of the file you want to link to, and the name of the new symbolic link. Here is an example:

$ ln -s /path/to/file /path/to/link

This creates a new symbolic link called link that points to the original file located at /path/to/file. Symbolic links are just files that contain the path to the original file, and they are treated like a shortcut or alias to the original file. If you modify the contents of the original file, the symbolic link will point to the updated version of the file.

It is advisable to use symbolic links when you want to create a reference to a file or directory in another location without duplicating the file's contents. This can be useful when you want to create shortcuts to frequently accessed files or when you want to reference files or directories that are located in different parts of the file system.

Symbolic links can also be used to simplify file paths, as well as to provide compatibility with legacy software or file systems that have different naming conventions or limitations.

1. **Compare hard and symbolic links?**

Hard links and symbolic links are two types of file system links that allow you to reference a file in a different location from its original location.

One key difference between hard links and symbolic links is that a hard link is a direct reference to the original file, while a symbolic link is a file that contains a path to the original file.

Another difference is that a hard link can only reference files that are located on the same file system, while a symbolic link can reference files on different file systems.

Here are some additional differences between the two:

Hard links are created using the ln command without any options, while symbolic links are created using the -s option: ln -s target link\_name.

A hard link can be thought of as a "clone" of the original file, since it shares the same inode (identifier for the file) as the original file. Symbolic links, on the other hand, are separate files that reference the original file.

If you delete the original file, a hard link will continue to function as a separate file that contains the same content as the original file. If you delete the original file that a symbolic link references, the link will be broken and will no longer function.

You can create multiple hard links to a file, and all the links will reference the same inode. With symbolic links, you can create multiple links to the same file, but each link will have its own inode and therefore be a separate file.

Overall, hard links are useful for creating multiple references to the same file within a single file system, while symbolic links are useful for creating references to files that are located on different file systems or for simplifying file paths.

**11. There is an original file and two links - symbolic and hard - have been created for it. What will happen with other files, if you delete:**

**- original file;**

**- symbolic link;**

**- hard link.**

If the original file has two links - symbolic and hard - and one of them is deleted, the original file will remain unchanged, but the remaining link will be broken. If the original file is deleted, both links will become broken, and the data stored in the file will be lost. However, if the file is still opened by a program, the program will continue to work with it, and the space occupied by the file will not be freed until the program closes the file.

If a symbolic link is deleted, the original file and other links to it will remain unchanged. However, if the symbolic link was the last link to the file, the file will become inaccessible, and its space will be freed.

If a hard link is deleted, the original file and other links to it will remain unchanged, and the data stored in the file will not be lost until all links to the file are deleted.

**Conclusion: I got hands-on experience with the Bash shell and the basics of changing file owners and permissions in Linux. You also learned about special directories and files in Linux, such as symbolic links and hard links to files, and how to use them.**