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

**Мета роботи:**

1. Отримання практичних навиків роботи з командною оболонкою Bash.
2. Знайомство з базовими діями при зміні власників файлів.
3. Знайомство з базовими діями при зміні прав доступу до файлів
4. Знайомство з спеціальними каталогами та файлами в Linux.

**Матеріальне забезпечення занять**

1. ЕОМ типу IBM PC.

2. ОС сімейства Windows (Windows 7).

3. Віртуальна машина – Virtual Box (Oracle).

4. Операційна система GNU/Linux – CentOS.

5. Сайт мережевої академії Cisco netacad.com та його онлайн курси по Linux

**Короткі теоретичні відомості:**

**Introduction to File ownership**

File ownership is critical for file security. Every file has a user owner and a group owner.

This chapter focuses on how to specify the user and group ownership of a file. In addition, the concept of file and directory permissions is explored, including how to change the permissions on files and directories. Default permissions are the permissions given to files and directories when they are initially created.

File Ownership

By default, users own the files that they create. While this ownership can be changed, this function requires administrative privileges. Although most commands usually show the user owner as a name, the operating system is associating the user ownership with the UID for that username.

Every file also has a group owner. By default, the primary group of the user who creates the file is the group owner of any new files. Users are allowed to change the group owner of files they own to any group that they belong to. Similar to user ownership, the association of a file with a group is not done internally by the operating system by name, but by the GID of the group.

Since ownership is determined by the UID and GID associated with a file, changing the UID of a user (or deleting the user) has the effect of making a file that was originally owned by that user have no real user owner. When there is no UID in the /etc/passwd file that matches the UID of the owner of the file, then the UID (the number) is displayed as the user owner of the file instead of the username (which no longer exists). The same occurs for groups.

The id command can be useful for verifying which user account you are using and which groups you have available to use. By viewing the output of this command, you can see the user's identity information expressed both as a number and as a name.

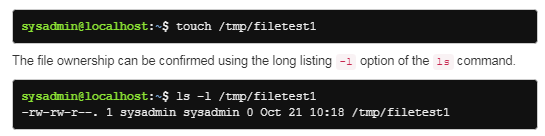
The output of the id command displays the UID and user account name of the current user followed by the GID and group name of the primary group and the GIDs and group names of all group memberships::



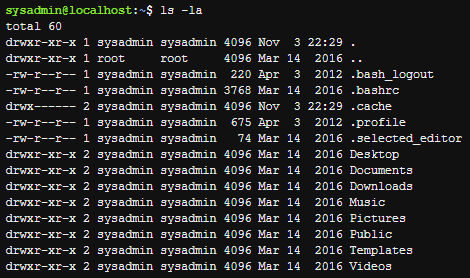
The above example shows the user has a UID of 1001 for the user account sysadmin. It also shows that the primary group for this user has a GID of 1001 for the group sysadmin.

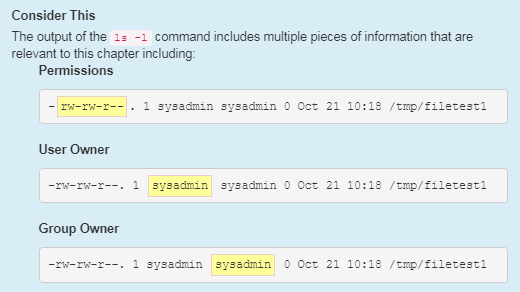
Because the user account and primary group account have the same numeric identifier and name, this indicates that this user is in a User Private Group (UPG). In addition, the user in this example belongs to four supplemental groups: the adm group with a GID of 4, the sudo group with a GID of 27, the research group with a GID of 1005 and the development group with a GID of 1006.

When a file is created, it belongs to the current user and their current primary group. If the user from the previous example executes the touch command to create a file, then the user owner of the file is the sysadmin user, and the group owner is the sysadmin group:



File ownership also applies to hidden files in the system. Hidden files, which begin with the period . character are listed using the -a option of the ls command. The first two hidden files listed are the current . and parent .. directories respectively. The ownership of all files and subdirectories within the current directory can be listed using the ls -la command.





Changing Groups

If you know that the file you are about to create should belong to a group different from your current primary group, then you can use the newgrp command to change your current primary group.

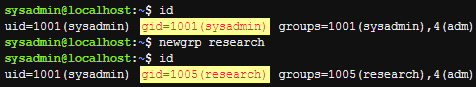
*newgrp group\_name*

The id command lists your identity information, including your group memberships. If you are only interested in knowing what groups you belong to, then you can execute the groups command:



The output of the groups command may not be as detailed as the output of the id command, but if all you need to know is what groups you can switch to by using the newgrp command, then the groups command provides the information that you need. The id command output does show your current primary group, so it is useful for verifying that the newgrp command succeeded.

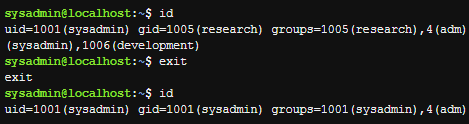
For example, if the sysadmin user was planning on having a file owned by the group research, but that wasn't the user's primary group, then the user could use the newgrp command and then verify the correct primary group with the id command before creating the new file:

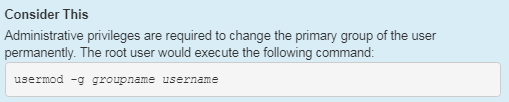


According to the output of the previous commands, initially the user's GID is 1001 for the sysadmin user, then the newgrp command is executed, and the user's primary GID becomes 1005, the research group. After these commands were executed, if the user were to create another file and view its details, the new file would be owned by the research group:



The newgrp command opens a new shell; as long as the user stays in that shell, the primary group won't change. To switch the primary group back to the original, the user can leave the new shell by running the exit command. For example:





**Introduction to Special Directories and Files**

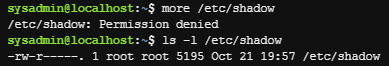
In most circumstances, the basic Linux permissions, read, write, and execute, are enough to accommodate the security needs of individual users or organizations.

However, when multiple users need to work routinely on the same directories and files, these permissions may not be enough. The special permissions setuid, setgid and the sticky bit are designed to address these concerns..

Setuid

When the setuid permission is set on an executable binary file (a program) the binary file is run as the owner of the file, not as the user who executed it. This permission is set on a handful of system utilities so that they can be run by normal users, but executed with the permissions of root, providing access to system files that the normal user doesn't normally have access to.

Consider the following scenario in which the user sysadmin attempts to view the contents of the /etc/shadow file:



The permissions on /etc/shadow do not allow normal users to view (or modify) the file. Since the file is owned by the root user, the administrator can temporarily modify the permissions to view or modify this file.

Now consider the passwd command. When this command runs, it modifies the /etc/shadow file, which seems impossible because other commands that the sysadmin user runs that try to access this file fail. So, why can the sysadmin user modify the /etc/shadow file while running the passwd command when normally this user has no access to the file?

The passwd command has the special setuid permission set. When the passwd command is run, and the command accesses the /etc/shadow file, the system acts as if the user accessing the file is the owner of the passwd command (the root user), not the user who is running the command.

You can see this permission set by running the ls -l command:



Notice the output of the ls command above; the setuid permission is represented by the s in the owner permissions where the execute permission would normally be represented. A lowercase s means that both the setuid and execute permission are set, while an uppercase S means that only setuid and not the user execute permission is set.

Like the read, write and execute permissions, special permissions can be set with the chmod command, using either the symbolic and octal methods.

To add the setuid permission symbolically, run:

*chmod u+s file*

To add the setuid permission numerically, add 4000 to the file's existing permissions (assume the file originally had 775 for its permission in the following example):

*chmod 4775 file*

To remove the setuid permission symbolically, run:

*chmod u-s file*

To remove the setuid permission numerically, subtract 4000 from the file's existing permissions:

*chmod 0775 file*

Previously, we set permission with the octal method using three-digit codes. When a three-digit code is provided, the chmod command assumes that the first digit before the three-digit code is 0. Only when four digits are specified is a special permission set.

If three digits are specified when changing the permission on a file that already has a special permission set, the first digit will be set to 0, and the special permission will be removed from the file.

Setgid

The setgid permission is similar to setuid, but it makes use of the group owner permissions. There are two forms of setgid permissions: setgid on a file and setgid on a directory. The behavior of setgid depends on whether it is set on a file or directory.

The setgid permission on a file is very similar to setuid; it allows a user to run an executable binary file in a manner that provides them additional (temporary) group access. The system allows the user running the command to effectively belong to the group that owns the file, but only in the setgid program.

A good example of the setgid permission on an executable file is the /usr/bin/wall command. Notice the permissions for this file as well as the group owner:



You can see that this file is setgid by the presence of the s in the group's execute position. Due to this executable being owned by the tty group, when a user executes this command, the command is able to access files that are group owned by the tty group.

This access is important because the /usr/bin/wall command sends messages to terminals, which is accomplished by writing data to files like the following:



Note that the tty group has write permission on the files above while users who are not in the tty group ("others") have no permissions on these files. Without the setgid permission, the /usr/bin/wall command would fail.

When set on a directory, the setgid permission causes files created in the directory to be owned by the group that owns the directory automatically. This behavior is contrary to how new file group ownership would normally function, as by default new files are group owned by the primary group of the user who created the file.

In addition, any directories created within a directory with the setgid permission set are not only owned by the group that owns the setgid directory, but the new directory automatically has setgid set on it as well. In other words, if a directory is setgid, then any directories created within that directory inherit the setgid permission.

By default when the ls command is executed on a directory, it outputs information on the files contained within the directory. To view information about the directory itself add the -d option. Used with the -l option, it can be used to determine if the setgid permission is set. The following example shows that the /tmp/data directory has the setgid permission set and that it is owned by the demo group.



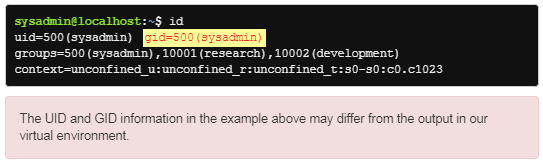
In a long listing, the setgid permission is represented by an s in the group execute position. A lowercase s means that both setgid and group execute permissions are set:

*drwxrwsrwx. 2 root demo 4096 Oct 30 23:20 /tmp/data*

An uppercase S means that only setgid and not group execute permission is set. If you see an uppercase S in the group execute position of the permissions, then it indicates that although the setgid permission is set, it is not really in effect because the group lacks the execute permission to use it:

*drwxrwSr-x. 2 root root 5036 Oct 30 23:22 /tmp/data2*

Typically files created by the user sysadmin are owned by their primary group, also called sysadmin.



However, if the user sysadmin creates a file in the /tmp/data directory, the setgid directory from the preceding example, the group ownership of the file isn't the sysadmin group, but rather the group that owns the directory demo:



Why would an administrator want to set up a setgid directory? First, consider the following user accounts:

The user bob is a member of the payroll group.

The user sue is a member of the staff group.

The user tim is a member of the acct group.

In this scenario, these three users need to work on a joint project. They approach the administrator to ask for a shared directory in which they can work together, but that no one else can access their files. The administrator does the following:

Creates a new group called team.

Adds bob, sue, and tim to the team group.

Makes a new directory called /home/team.

Makes the group owner of the /home/team directory be the team group.

Gives the /home/team directory the following permissions: rwxrwx---

As a result, bob, sue, and tim can access the /home/team directory and add files. However, there is a potential problem: when bob creates a file in the /home/team directory, the new file is owned by his primary group:

*-rw-r-----. 1 bob payroll 100 Oct 30 23:21 /home/team/file.txt*

Unfortunately, while sue and tim can access the /home/team directory, they can't do anything with bob's file. Their permissions for that file are the others permissions (---).

If the administrator sets the setgid permission to the /home/team directory, then when bob creates a file, it is owned the team group:

*-rw-r-----. 1 bob team 100 Oct 30 23:21 /home/team/file.txt*

As a result, sue and tim would have access to the file through the group owner permissions (r--).

Certainly, bob could change the group ownership or the others permissions after creating the file, but that would become tedious if there were many new files being created. The setgid permission makes it easier for this situation.

**Made by Volodymyr Malamuzh**

**Завдання для попередньої підготовки.**

1. На базі розглянутого матеріалу дайте відповіді на наступні питання:
   1. Яке призначення команди id?

*The 'id' command in Linux is used to display information about the user ID (UID) and group ID (GID) of the current user and their membership in other groups.*

* 1. Як переглянути які права доступу має власник файлу?

*To view the file owner's permissions on Linux, use the 'ls' command with the '-l' switch*.

* 1. Як змінити власника групи?

*To change the owner of a group on Linux, use the 'chgrp' command.*

* 1. Як можна переглянути у терміналі який тип поточного файлу? Наведіть приклади для різних типів файлів

*To view the type of the current file in the terminal, you can use the file command. This command allows you to determine the type of file based on its contents and other attributes.*

* *For a text file:*

*$ file sample.txt*

* *For an executable file:*

*$ file myprogram*

* *For an image:*

*$ file image.jpg*

* 1. Для чого використовуються дозволи Setuid та Setgid?

*The Setuid (set user ID) and Setgid (set group ID) permissions are special permission bits in the Linux system that are used to grant temporary privileges to the user who runs the executable.*

*The Setuid permission is set on an executable file and gives the user running the file the temporary right to execute the file with the rights of the file owner.*

*The Setgid permission works in a similar way, but gives the user who runs the executable temporary rights to execute the file with the rights of the group to which the executable belongs.*

* 1. Для чого в системі потрібен так званий “липкий біт” (Sticky Bit). Наведіть приклади коли цей дозвіл доцільно використовувати.

*Sticky Bit is a special permission that can be set for folders in Unix-like operating systems such as Linux. Sticky Bit allows you to restrict access to a folder only to the owners of the files contained in that folder and administrators.*

*One example of a sticky bit is the /tmp folder. Normally, any user of the system can create, modify, and delete files from this folder. But if you set a sticky bit for this folder, then the user can delete only his own files, and only the system administrator can delete the files of other users.*

*The sticky bit can also be used to protect important files used in the system from accidental deletion by users.*

**Хід роботи.**

* 1. Опрацюйте всі приклади команд, що представлені у лабораторних роботах курсу ***NDG Linux Essentials:***
* ***Lab 17: Ownership and Permissions***
* ***Lab 18: Special Directories and Files***
  1. Створіть таблицю команд вивчених у п.2 ходу роботи у наступному вигляді:

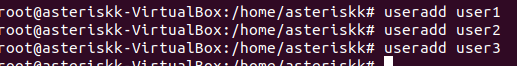
| Назва команди | Її призначення та функціональність |
| --- | --- |
| id | The output command displays the UID and user account name of the current user followed by the GID and group name of the primary group and the GIDs and group names of all group memberships |
| ls -l | To confirm ownership |
| ls -la | Lists the owners of all files and subdirectories in the current director |
| newgrp | to change your current primary group. |
| groups | Shows information about the groups you belong to |
| Chgrp | To change the group owner of an existing file |
| stat | displays more detailed information about a file, including providing the group ownership both by group name and GID number |
| chown | allows the root user to change the user ownership of files and directories. |
| chmod | is used to change permissions on files and directories. |
| umask | is used to determine default permissions that are set when a file or directory is created. |
| Ls -i | displays the inode number of a file. |
| ln | is used with two arguments. The first argument is an existing file name to link to, called a target, and the second argument is the new file name to link to the target. |

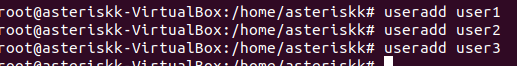
\*\*\***Скріншоти** виконання команд в терміналі можна **не представляти**, достатньо **коротко описати команди в таблиці**.

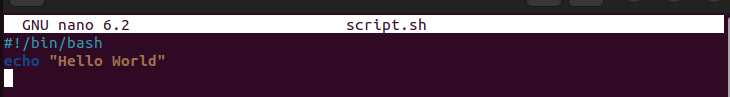
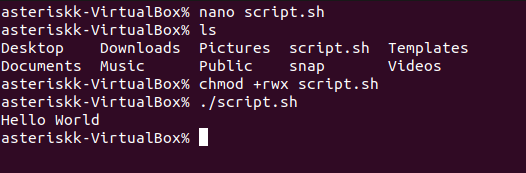
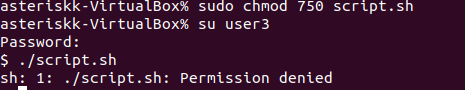
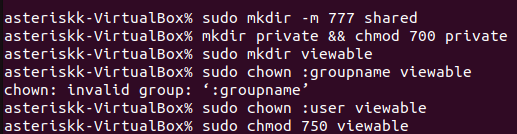
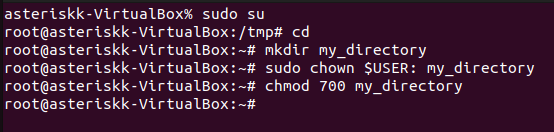
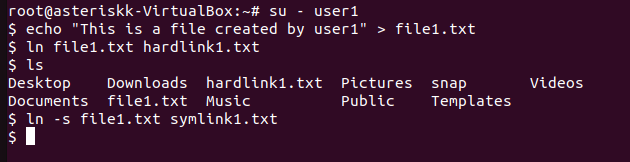
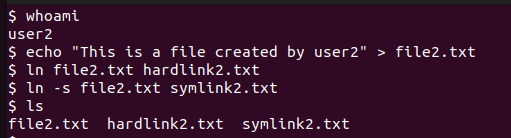
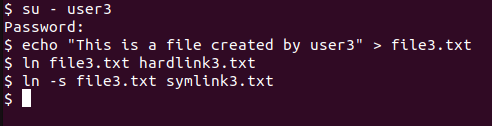
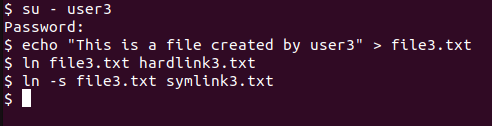
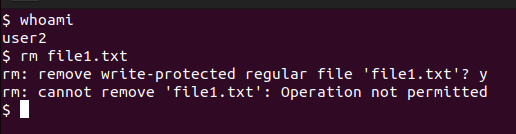
**Made by Gennadiy Rumyantsev**

* 1. Виконайте наступні практичні завдання у терміналі наступні дії (продемонструвати скріншоти):

Prepared by Rumiantsev Hennadiy

* створіть трьох нових користувачів;
* 
* створіть нову групу користувачів, туди додайте двох, з трьох створених користувачів;



* створіть новий файл, який буде доступний на зчитування, редагування та виконання власником файлу, наприклад найпростіший скриптовий сценарій;
* 
* 
* для користувачів групи власника надайте дозволи на перегляд та виконання (без дозволу на редагування) цього файлу;
* 
* для інших користувачів заборонити доступ до цього файлу;
* 
* подібні дії виконайте для директорій - створіть директорію, яка буде доступна для всіх трьох користувачів, створіть директорію, яку буде доступна тільки для власника, створіть директорію, яку користувачі групи власника зможуть переглядати, але не редагувати;
* 
* створіть порожній файл під назвою emptyfile за допомогою команди touch emptyfile. Тепер “обнуліть” дозволи для файлу з chmod 000 emptyfile. Що станеться, якщо змінити дозволи для emptyfile, передавши лише одне значення для chmod у числовому режимі, наприклад, chmod 4 emptyfile? Що буде, якщо ми використаємо два числа, наприклад chmod 44 emptyfile? Що ми можемо дізнатися про те, як chmod зчитує числове значення?
* After executing the chmod 000 emptyfile command, the file "emptyfile" will have all permissions reset to zero. This means that any user or program will not be allowed to read, write, or execute this file.
* If you run the chmod 4 emptyfile command, it will only set read permission for the owner of the file. That is, the permissions for the group and other users will remain unchanged.
* The chmod 44 emptyfile command will set read permission for the owner and group of the file, but deny access to other users.
* To learn how chmod reads a numeric value, you can look at the three numbers used in the chmod command. The first number indicates the permissions for the owner of the file, the second for the group to which the file belongs, and the third for all other users. Each number consists of three digits, where each digit is responsible for a specific permission (read, write, or execute). For example, the number 777 in the chmod 777 file command means that all users are granted all possible permissions for this file
* створіть каталог під назвою, де всі файли автоматично будуть належати Вашій групі користувачів і можуть бути видалені лише користувачем, який їх створив?
* 
* під кожним користувачем створіть по одному новому файлу, та жорстке та символічне посилання на нього;
* 
* 
* 
* спробуйте іншими користувачами переглянути ці файли;
* 
* спробуйте іншими користувачами видалити ці файли, зробіть висновки.
* 

Сonclusion:

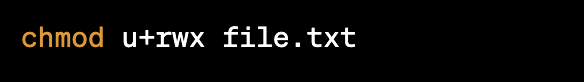
This hands-on demonstrated how to create new users and user groups in Linux, how to provide different levels of access to files and directories for different users and user groups, and how to create hard and symbolic links to files

**Made by Anton Khomenko**

**Контрольні запитання**

1. Наведіть приклади зміни прав доступу символічним методом (Symbolic Method)?

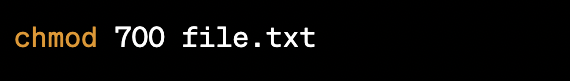
Changing access rights using the symbolic method is done using the chmod command on the command line in Unix-like operating systems.



This command adds read, write, and execute permissions to the file.txt file for the file owner.

1. Наведіть приклади зміни прав доступу числовим методом (numeric method, octal method)?

Changing permissions using the numeric method is done with the chmod command on the command line in Linux, using numbers to represent permissions.

Changes the read, write, and execute permissions for the owner of a file:

This command sets the read (r), write (w), and execute (x) permissions for the owner(s) of file.txt, where "7" represents the owner's combination of permissions (rwx).

1. Чи можна виконати файл, для якого є права на виконання, але не встановлені права на читання (--x)? Поясніть.

Yes, on Linux, you can execute a file that has execute privileges but not read privileges (--x).

This is because the execute (x) permission allows you to execute the file, i.e. to run its contents as a program or script. Read permissions (r) allow you to read the contents of a file. However, when read permissions are not set and execute permissions are set, you can still execute the file, but you cannot read its contents.

This is because executing a file does not require reading its contents. When you execute a file, the operating system executes its contents as a program or script, without having to open the file for reading. The contents of the file are read directly into the operating system's memory and executed as executable code.

So, if a file has runtime permissions but no read permissions, you can still execute it as a program or script, but you won't be able to read its contents. This can be used to protect the contents of a file from unauthorised access by allowing the file to be used as executable code without being readable.

1. Яке призначення команди umask?

The purpose of the umask command is to set the default permissions for new files and directories that the user creates. It subtracts the permissions bits from the maximum value (typically 777 for files and 666 for directories) to set the permissions mask that is used to restrict the permissions of newly created files and directories.

The umask command allows the user to set a default permissions mask to suit their needs and security requirements. It is typically used in scripts or in shell settings such as .bashrc or .bash\_profile to set default permissions when creating new files and directories.

1. Якщо ми змінюємо права доступу та дозволи в поточній сесії чи будуть вони збережені в наступній?

No, changes to access rights and permissions made in the current Linux session are not automatically saved to subsequent sessions. Each Linux session is separate and does not save changes to permissions and access rights for subsequent sessions.

If you change access rights or permissions for files or directories during the current session, these changes will be applied only to the current session. The next time you log in or open a new shell, the changes will not be saved and the access rights will be restored to the state set in the default settings.

If you need to change access rights or permissions so that they are applied the next time you log in, you should make changes to the default settings, such as the /etc/fstab file system or shell configuration files (for example, .bashrc or .bash\_profile), or use the umask command in scripts that run at login.

1. Чи є якийсь шаблон, яким система користується щодо прав та доступів при створенні нових файлів. Як можна змінити права дозволу за замовчуванням?

Yes, Linux has a template that the system uses when creating new files and directories. This pattern is called "umask" and it defines the permissions that are set by default for new files and directories.

Conditionally, umask defines which permissions will not be set by default for new files and directories. This means that if a permissions bit is included in umask, it will not be included in the permissions of the new file or directory. The umask value is measured in octal notation, where each position represents the corresponding permissions bit (for example, 7 corresponds to rwx permissions, 6 corresponds to rw- permissions, 0 corresponds to --- permissions).

The umask value can be viewed using the umask command in the terminal. For example, if you enter umask without arguments, you will be shown the current umask value.

To change the default umask, you can use the umask command with the new value in your shell configuration file (for example, .bashrc or .bash\_profile). For example, if you set umask 022, the default permissions for new files and directories will be set to rw-r--r-- (i.e. rwx for owner and rw- for group and others). It is important to consider the inheritance of permissions from the parent directory, so setting umask to a high level may result in less restrictive permissions for new files and directories.

1. Уявіть, що програмі потрібно створити одноразовий тимчасовий файл, який більше ніколи не знадобиться після закриття програми. Який правильний каталог для створення цього файлу?

You can use the /tmp directory on Linux to create a one-time temporary file that is no longer needed after the application is closed. The /tmp directory is for temporary files that can be used by different programs on the system. It usually has high permissions (for example, rwxrwxrwt), which allows you to create files with any permissions in this directory.

However, it is worth noting that files created in the /tmp directory can be automatically deleted when the system is rebooted or as part of the /tmp cleanup process. Therefore, if you need to save files for a long time, it is better to use other directories with appropriate access rights.

If your application requires the creation of reusable temporary files, it may be wiser to create your own directory for these files with appropriate permissions instead of using the /tmp directory.

1. Яким чином можна створити жорстке посилання? В яких ситуаціях їх доцільно використовувати?

A hard link, also known as a hardlink, is a relative link to a file or directory in the Linux file system. A hard link creates a reference to the contents of a file or directory without creating a separate node in the file system. This means that the contents of a file or directory can be accessed from multiple paths, and changes made in one path will be visible in all other paths pointing to the same file or directory.

To create a hard link, you can use the ln command.

1. Яким чином можна створити символічне посилання? В яких ситуаціях їх доцільно використовувати?

Symbolic links, also known as softlinks or symlinks, are references to files or directories in the Linux file system that contain the absolute or relative path to the original file or directory. Unlike hard links, symbolic links create a separate node in the file system that points to the path to the original file or directory.

To create a symbolic link, you can use the ln command with the -s option.

1. Порівняйте жорсткі та символічні посилання?

Hard links:

-These are created at the file system level and point to the same data node as the original file or directory.

-They have the same inode (node index) as the original file or directory.

-They cannot point to directories or files on different file systems or network shares.

-They cannot be used to create links to directories.

-If you delete the original file or directory, the hard link remains, but it's no longer available for use.

Symbolic links:

-They create a separate node in the file system that points to the path of the original file or directory.

-They have their own inode and can point to files or directories on different file systems or network shares.

-They can be used to create links to directories.

-If you delete the original file or directory, the symbolic link will remain, but it will become broken, meaning that it points to a non-existent file or directory.

1. Є файл оригінал та для нього створено два посилання - символічне та жорстке. Що відбудеться з іншими файлами, якщо видалити:

* файл оригінал;
* символічне посилання;
* жорстке посилання.

If you delete the original file, any references to that file, whether symbolic or hard, will remain, but they will no longer be available for use. However, they will remain in place and you can restore the file by creating a new file with the same name or restoring the original file from a backup.

Deleting a symbolic link does not affect the original file or any other symbolic or hard links, as symbolic links simply point to the path to the original file and have no physical dependency on it.

If you delete a hard link, the original file and any other hard links to that file remain unchanged because they point to the same data node on the file system. However, if you delete the original file, it will be lost, even if the hard link remains in place. This can result in a "broken" hard link pointing to a non-existent file.

Сonclusion:

*Changing file owners and permissions are key aspects of security in Linux. Thanks to special directories and files such as /etc/passwd and /etc/group, users can effectively control access to various files and directories.*