“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”**

Виконав студент

групи РПЗ-23А

Туровський В.

Перевірила викладач

Сушанова В.С.

Київ 2025

**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, .прав доступу до файлів
3. Знайомство з спеціальними каталогами та файлами в Linux.

**Матеріальне забезпечення занять:**

1. ЕОМ типу IBM PC.

2. ОС сімейства Windows та віртуальна машина Virtual Box (Oracle).

3. ОС GNU/Linux (будь-який дистрибутив).

4. 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.

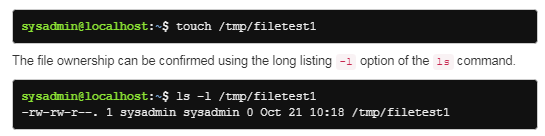
File Ownership

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.

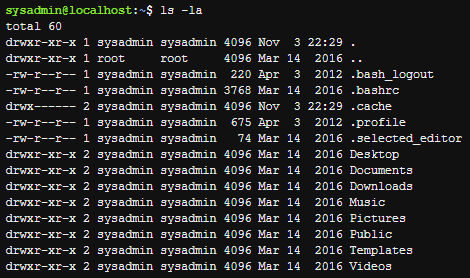
The id command can be useful for verifying which user account you are using and which groups you have available to use. By viewing the output of this command, you can see the user's identity information expressed both as a number and as a name. The output of the id command displays the UID and user account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

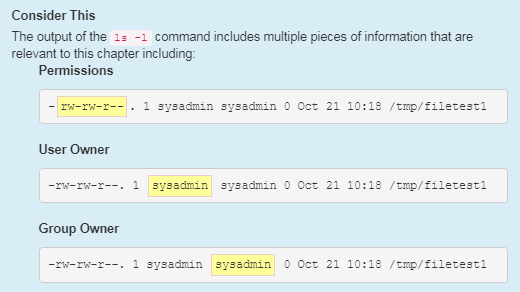


The above example shows the user has a UID of 1001 for the user account sysadmin. It also shows that the primary group for this user has a GID of 1001 for the group sysadmin. Because the user account and primary group account have the same numeric identifier and name, this indicates that this user is 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.

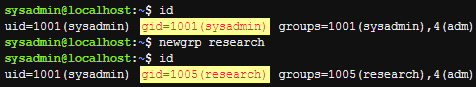
*newgrp group\_name*

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.

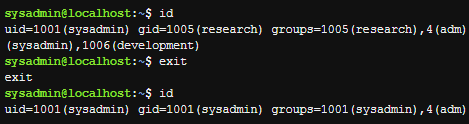
For example, if the sysadmin user was planning on having a file owned by the group research, but that wasn't the user's primary group, then the user could use the newgrp command and then verify the correct primary group with the id command before creating the new file:

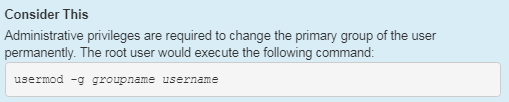


According to the output of the previous commands, initially the user's GID is 1001 for the sysadmin user, then the newgrp command is executed, and the user's primary GID becomes 1005, the research group. After these commands were executed, if the user were to create another file and view its details,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

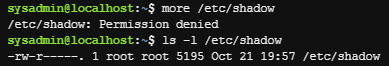
In most circumstances, the basic Linux permissions, read, write, and execute, are enough to accommodate the security needs of individual users or organizations.

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.



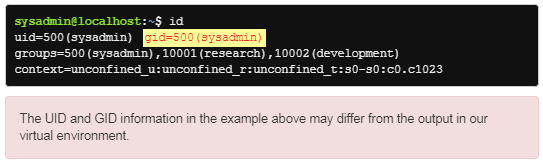
In a long listing, the setgid permission is represented by an s in the group execute position. A lowercase s means that both setgid and group execute permissions are set:

*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.

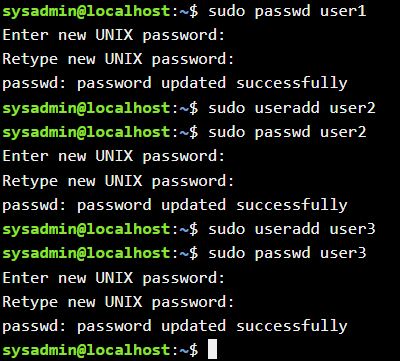
**Хід роботи:**

* 1. Початкова робота в CLI-режимі в Linux ОС сімейства Linux:
  2. Запустіть операційну систему Linux Ubuntu. Виконайте вхід в систему та запустіть термінал ***(якщо виконуєте ЛР у 401 ауд.)***.
  3. Запустіть віртуальну машину Ubuntu\_PC ***(якщо виконуєте завдання ЛР через академію netacad)***
  4. Запустіть свою операційну систему сімейства Linux ***(якщо працюєте на власному ПК та її встановили)*** та запустіть термінал.
  5. Опрацюйте всі приклади команд, що представлені у лабораторних роботах курсу ***NDG Linux Essentials: Lab 17: Ownership and Permissions*** та ***Lab 18: Special Directories and Files.*** Створіть таблицю команд вивчених у п.2 ходу роботи у наступному вигляді:

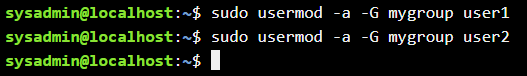
| Назва команди | Її призначення та функціональність |
| --- | --- |
|  |  |
|  |  |

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

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

nano [myscript.sh](http://myscript.sh)

#!/bin/bash echo 'Привіт від скрипта!'

chmod u+x [myscript.sh](http://myscript.sh)

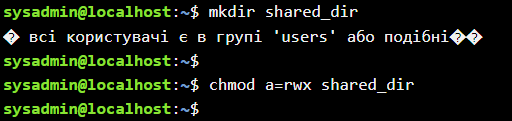
* для користувачів групи власника надайте дозволи на перегляд та виконання (без дозволу на редагування) цього файлу;

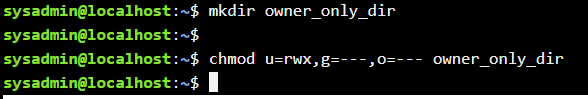


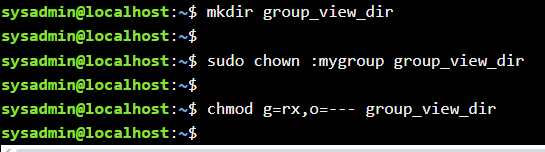
* для інших користувачів заборонити доступ до цього файлу;



* \*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?

**Команда chmod та числові режими:**

Bash

touch emptyfile

chmod 000 emptyfile

Файл emptyfile тепер не має жодних прав для жодного користувача.

Bash

chmod 4 emptyfile

Це еквівалентно chmod u=r,g=---,o=--- emptyfile. Оскільки вказано лише одне число, воно інтерпретується як права для власника. Решта прав (для групи та інших) залишаються без змін (в даному випадку ---).

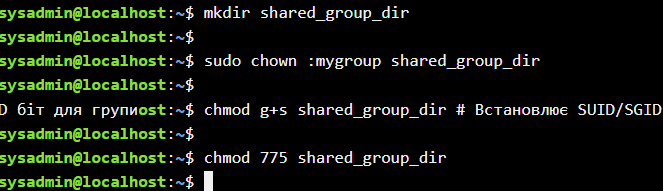
Bash

chmod 44 emptyfile

Це призведе до помилки, оскільки числовий режим chmod очікує три цифри (власник, група, інші).

Числовий режим chmod використовує тризначне вісімкове число, де кожна цифра представляє права для власника, групи та інших відповідно. Якщо передається менше трьох цифр, поведінка може бути непередбачуваною або призвести до помилки. Одна цифра інтерпретується як права лише для власника, залишаючи права для групи та інших без змін.

* \*\*створіть каталог під назвою, де всі файли автоматично будуть належати Вашій групі користувачів і можуть бути видалені лише користувачем, який їх створив?



У цій директорії всі нові файли автоматично успадкують групу mygroup. Однак, стандартні дозволи 775 дозволяють членам групи mygroup записувати та видаляти файли, створені іншими членами цієї групи. Щоб обмежити видалення лише власником файлу, потрібні складніші налаштування, можливо, з використанням ACL (Access Control Lists), що виходить за рамки стандартних прав доступу.

* \*\*під кожним користувачем створіть по одному новому файлу, та жорстке та символічне посилання на нього;

# Під user1

su user1

mkdir user1\_files

cd user1\_files

touch file1.txt

ln file1.txt hardlink1.txt

ln -s file1.txt symlink1.txt

exit

# Під user2

su user2

mkdir user2\_files

cd user2\_files

touch file2.txt

ln file2.txt hardlink2.txt

ln -s file2.txt symlink2.txt

exit

# Під user3

su user3

mkdir user3\_files

cd user3\_files

touch file3.txt

ln file3.txt hardlink3.txt

ln -s file3.txt symlink3.txt

exit

* \*\*спробуйте іншими користувачами переглянути ці файли;

su user2

cat /home/user1/user1\_files/file1.txt

ls -l /home/user1/user1\_files/file1.txt /home/user1/user1\_files/hardlink1.txt /home/user1/user1\_files/symlink1.txt

exit

За замовчуванням, якщо немає спеціальних обмежень на права доступу до домашнього каталогу user1 та файлів у ньому, user2 зможе переглянути вміст file1.txt (якщо є права на читання для "інших").

* \*\*спробуйте іншими користувачами видалити ці файли, зробіть висновки.

su user2

rm /home/user1/user1\_files/file1.txt

rm /home/user1/user1\_files/hardlink1.txt

rm /home/user1/user1\_files/symlink1.txt

exit

**Висновки:** Можливість переглядати файли іншими користувачами залежить від прав доступу до файлу та каталогу, в якому він знаходиться. За замовчуванням, права "інших" часто дозволяють читання. Можливість видаляти файли, створені іншими користувачами, в каталозі, що належить іншому користувачеві, зазвичай заборонена, навіть якщо є права на запис до цього каталогу (якщо не встановлено sticky bit на каталозі).

Щоб дозволити членам групи mygroup переглядати файли один одного, потрібно буде змінити права доступу до файлів, створених user1 та user2, надавши права на читання для групи mygroup.