

Users, Groups & Permissions

Session 9

Users, Groups & Permissions

Session 9

Objectives:

- Understand file and directory permissions (rwx, ls -l)
- Use chmod (symbolic and numeric modes)
- Modify ownership using chown and chgrp
- Understand how users and groups are stored in the system
- Create, delete, and modify users and groups
- Use sudo and su to escalate privileges

Password

🔧 Linux Login and Password Authentication:

- To enter a UNIX/Linux system, you need an **account**, a **username**, and a **password**



An account represents a **unique user identity** on the system. It contains all the user-related information and permissions.

A **username** is a **human-readable identifier** for the user account. **username** helps the system recognize which **account** you are trying to access.

A **password** is used for **authentication** — it ensures that only **authorized users** can access the system or their specific account.

Password

- Open `/etc/passwd`. What do you see?



Password

- Open `/etc/passwd`

`username:password:UID:GID:GECOS:home_directory:shell`



- **username**: The name of the user.
- **password**: A placeholder for the user's password (usually an 'x' or '*' if the password is stored elsewhere, like `/etc/shadow`).
- **UID**: User ID — a unique numeric identifier for the user.
- **GID**: Group ID — a numeric identifier for the user's primary group.
- **GECOS**: Optional information, often containing the real name or a description of the user (can be empty).
- **home_directory**: The path to the user's home directory.
- **shell**: The path to the user's default shell (e.g., `/bin/bash`).

Modify password

- Use the command **passwd**:

passwd <username>



- Adding a New User using **useradd**

useradd <username>

You can add also some information:

useradd -m -s /bin/bash thomas_dejaeger

- m: Creates the home directory for the user.
- s: Specifies the shell to use (e.g., /bin/bash)

Understanding File Permissions

Each file or directory has:

- **A type:** - for regular file, d for directory
- **A set of permissions:**
 - 3 for the owner (user): The owner of the file
 - 3 for the group: Members of the file's group
 - 3 for others: Everyone else (not user or group)

Ex: `ls -l`

Understanding File Permissions

`-rwxr-xr--`

First char: file type

Next 3: user permissions

Next 3: group permissions

Last 3: others

r = read

w = write

x = execute

Understanding File Permissions

-rwxr-xr--

- = regular file

rwx = user: read, write, execute

r-x = group : read, execute

r-- = other: read only

Symbolic chmod (change permissions)

There's a utility called **chmod**, which can modify permissions on files:

Legend:

- u = user
- g = group
- o = other
- a = all
- + = add, - = remove, = = set

Examples:

- `chmod u+xw file` # add execute & write for user
- `chmod g-w file` # remove write for group
- `chmod o=r file` # set other to read only

Numeric chmod and Permission Calculation

Use numbers from 0–7:

Permission	Value
r	4
w	2
x	1

For example:

`chmod 755 file` → `rwxr-xr-x`

`chmod 644 file` → `rw-r--r--`

7 = `rw`x, 5 = `r`-x, 4 = `r`--

chmod Demo (Before & After)

```
touch demo.txt
```

```
ls -l demo.txt
```

```
chmod 600 demo.txt
```

```
ls -l demo.txt
```

Output:


```
-rw----- 1 user user 0 demo.txt
```

Special Permissions: setuid, setgid, and Sticky Bit

What are Special Permissions?

Beyond basic rwx, Linux supports 3 special permission bits:

Bit	Applies To	Effect
setuid	Executable files	Run as the file's owner, not the executing user
setgid	Executable files or directories	Run as file's group (or inherit group on files)
sticky	Directories	Only file owner can delete, even if others have write access

Symbol	Meaning	Example
s	setuid or setgid is active and file is executable	-rwsr-xr-x (setuid on file) drwxrwsr-x (setgid on dir)
S	setuid or setgid is set, but file is not executable →  misconfigured	-rwSr--r-- or -rw-r-Sr--
t	Sticky bit is set (only owner can delete files)	drwxrwxrwt (e.g. /tmp)

Misusing them can open serious security holes — always review `ls -l` carefully before setting them

Default Permissions and umask

What is umask?

When you create a new file or directory, it starts with default permissions, and **umask** determines what gets subtracted from those defaults

By default:

Type	Default Mode	Meaning
Files	666	rw-rw-rw-
Directories	777	rwxrwxrwx

Default Permissions and umask

What is umask?

When you create a new file or directory, it starts with default permissions, and **umask** determines what gets subtracted from those defaults

By default:

Type	Default Mode	Meaning
Files	666	rw-rw-rw-
Directories	777	rwxrwxrwx

The **umask** value subtracts permissions from the default
It's a **4-digit** octal number (e.g., 0022 or 0077)

Default Permissions and umask

Digit position What it means

0 (1st) Special bits: setuid, setgid, sticky (rarely used in umask)

0 (2nd) Permissions removed from user (owner)

2 (3rd) Permissions removed from group

2 (4th) Permissions removed from others

What is the first digit?

It's a convention in Unix to write file modes and masks in 4-digit octal (base 8)

Some tools (like install, umask, chmod) expect or display it that way

Helps avoid confusion when using special bits like setuid

Default Permissions and umask

Type: umask

Output: 0022

Create a file:
touch myfile.txt
ls -l myfile.txt

Output:
-rw-r--r-- Why?

Default: 666 (rw-rw-rw-) -(0022) -> 644 (rw-r--r--)

Default Permissions and umask

```
umask 0077  
touch private.txt  
ls -l private.txt
```

Output:

```
-rw----- Why?
```

Default: 666 (rw-rw-rw-) –(0077) -> 600 (rw-----)

Default Permissions and umask



IMPORTANT

Do not put random number for umask!!

For example I decided to type:

umask 0111

Create a folder

mkdir testdir

ls -ld testdir

What I will obtain?

Default Permissions and umask



IMPORTANT

Do not put random number for umask!!

For example I decided to type:

umask 0111

Create a folder

mkdir testdir

ls -ld testdir

What I will obtain?

777-111=666 -> drw-rw-rw-

No x!!! So you can not cd to this directory

cd: testdir: Permission denied

Solution?

Default Permissions and umask



IMPORTANT

Do not put random number for umask!!

For example I decided to type:

umask 0111

Create a folder

mkdir testdir

ls -ld testdir

What I will obtain?

777-111=666 -> drw-rw-rw-

No x!!! So you can not cd to this directory

cd: testdir: Permission denied

Solution

chmod +x testdir

ls -ld testdir

drwxrwxrwx

Ownership: chown and chgrp

Change file ownership with chown

```
ls -l file.txt
```

```
➔ -rw-r--r-- 1 alice editors 1234 May 2 14:10 file.txt
```

alice: owner user

editors=owner group

#Change user

```
chown newuser file.txt
```

#Change both, user and group

```
chown newuser:newgroup file.txt # change both
```

Change only file group using chgrp

```
chgrp newgroup file.txt
```

Ex:

```
touch demo.txt
```

```
sudo chown alice:editors demo.txt
```

```
ls -l demo.txt
```

Notes:

You need root privileges
(sudo) to change ownership

Only the owner or root can
change group (depending on
group membership)

Users and Groups: How Linux Stores Them

How does Linux manage users?

Linux stores user and group information in text-based system files. These are read by commands like `id`, `whoami`, `login`, `passwd`, etc.

File	Purpose	Example Line
<code>/etc/passwd</code>	Stores user accounts (username, UID, shell)	<code>alice:x:1001:1001:Alice:/home/alice:/bin/bash</code>
<code>/etc/shadow</code>	Stores encrypted passwords (restricted access)	<code>alice:\$6\$...:19000:0:99999:7:::</code>
<code>/etc/group</code>	Stores group info (name, GID, members)	<code>team:x:1002:bob,carol</code>

Useful commands:

<code>whoami</code>	# Print current username
<code>id</code>	# UID, GID, and all groups of current user
<code>groups</code>	# All groups of current user

Exemples

```
id bob -> uid=1002(bob) gid=1002(bob) groups=1002(bob),1003(team)
groups bob -> Show all groups user 'bob' belongs to
```

User and Group Management

What can you manage?

As a sysadmin (or a sudoer), you can:

- Create users and assign them to groups
- Set or change passwords
- Modify user properties (home, shell, groups)
- Remove users or groups

Examples:

- Create user

```
sudo useradd bob
```

- Set user's password

```
sudo passwd bob
```

- Create new group

```
sudo groupadd team
```


User and Group Management

Other examples:

(remember, with id bob:

`uid=1002(bob) gid=1002(bob) groups=1002(bob))`

- **Add User to a Secondary Group**

`sudo usermod -aG team bob`

-a = append (Don't overwrite)

-G = specify secondary group(s)

Check: `groups bob -> bob : bob team`

BE CAREFUL: `sudo usermod -G team bob` **OVERWRITES ALL THE GROUP MEMBERSHIPS**

- **Delete User or Group**

`sudo userdel bob` (if you want to remove also home directory add option `-r`)

`sudo groupdel team`

SU vs SUDO

su: Switch User (full session):

- You become root until you exit
- Requires the root user's password

You can switch to another user too: su Thomas, if you type whoiam the answer will be Thomas

sudo: Run One Command as Another User

- Executes one command as root
- Asks for your password, not root's
- Logs the command for security

Ex. sudo apt update

SU vs SUDO

Which should you use?

Use su when...

- You want a full root shell
- You know the root password
- You're in rescue/recovery mode

Use sudo when...

- You just need to run one admin command
- You're in the sudo group
- You're on a multi-user secure system

Add a user to the sudoers group:

```
sudo usermod -aG sudo bob
```

Controlling sudo Access with /etc/sudoers

Use sudo visudo (It checks for syntax errors and prevents you from breaking sudo (which could lock you out!))

Format:

username ALL=(ALL:ALL) ALL

Part	Meaning
username	The user allowed to use sudo
ALL	From any host (on multi-host setups, usually just ALL)
(ALL:ALL)	Can run commands as any user and as any group
ALL (last)	Can run any command

bob ALL=(root) /usr/bin/apt, /bin/mkdir, bob will be able to run sudo only for apt and mkdir command