Introduction to Linux: **Basic Permissions**

Philipp Fruck & Pius Walter

Users and Permissions

Goal of this lecture

- Understand how to manage users/groups on Linux
- Understand how Linux handles permissions
- Understand potential security risks of wrong file permissions

Why?

- In Linux "everything is a file", so we have to care about the files permission!

Switching users

Command	Description	Useful Flags / Examples
`su`	Switch shell to another user	
`sudo`	**Su**peruser **do**...	`sudo mycommand` run command as root
	Substitute **u**ser **do**...	`sudo -i` open interactive root shell
		`sudo -u sshd id`

In most cases you will want to become the root user.

Many ways to elevate: `sudo -i`, `sudo su`, `su root`.

> Note: su root requires your root password while sudo -i requires your sudo password!

Group Management

Command	Description	Useful Flags / Examples
`id`	Show user/group IDs	
`groups`	Show groups you belong to	
`groupadd` / `groupdel`	Add/delete a group	`groupadd --system groupname`
`groupmod`	Modify a group	

Each group is assigned a unique group ID (GID):

- `0`: The `root` group
- `1-999`: System groups
- `>=1000`: User groups

For more restrictions, see `grep GID /etc/login.defs` and `cat /proc/sys/kernel/ngroups_max`

User Management

Command	Description	Useful Flags / Examples
`whoami`	Show current user	
`who`	Show logged-in users	
`useradd` / `userdel`	Add/delete a user	`useradd --no-create-home service`
`usermod`	Modify a user	`usermod -aG secondarygroup myuser`
	(Re-authenticate after changes)	`usermod -g primarygroup myuser`
`adduser` / `deluser`	Add/delete a user interactively	Convenience wrappers

Similar to group management: user ID (UID) `0` is `root`, `1-999` are system users and `>=1000` are regular users.

Further restrictions: `grep UID /etc/login.defs`

Exercise

A simple recap:

1. Create user `foo`
2. Create group `bar`
3. Add user `foo` to group `bar`

Storing User Information

There are multiple files storing information about users/groups. One example is the `/etc/passwd`:

```
alice:x:1000:1000:Alice:/home/alice:/bin/bash
```

Field	Value
Username	alice
Password	x
UID	1000
GID	1000
GECOS	Alice
Home	/home/alice
Shell	/bin/bash

Originally, the second field contained the hashed password, but for user ID lookups to work `/etc/passwd` must be readable by all users!

Solution: `/etc/shadow`, only readable by root.

Group information is stored in `/etc/group` and `/etc/gshadow`. See also `man 5 shadow` / `man 5 group` / `man 5 passwd`.

Setting `/sbin/nologin` as the users default shell disabled login for a certain user (at least interactively).

Advanced: Try `sudo su daemon` vs `sudo -u daemon bash`.

File Permissions & Ownership

Permissions control **who** can **read**, **write**, or **execute** files

Each file/directory has permissions for:

- Owner (user)
- Group (collection of users)
- Other (everyone else)

Three basic permission types:

- `r` = read
- `w` = write
- `x` = execute (or enter directory)

Permission Model

We already know how to display permissions. Now let's learn how to interpret them.

```
$ ls -l
-rwxr-xr--  1 alice staff 1234 Oct  5 10:00 script.sh
# perms links owner group size date & time  filename
```

Q: Who has which permissions on the above script?

- Owner (`alice`) can read, write, execute
- Group (`staff`) can read, execute
- Everyone else can only read

Understanding Permissions

What does `-rwxr-xr--` mean?

- First character:
  - `-` = file,
  - `d` = directory,
  - `l` = symlink
- Next 9 characters (permissions):
  - User (owner): `rwx`
  - Group: `r-x`
  - Other: `r--`

Changing Permissions

Symbolic Mode

- `+` -> add
- `-` -> remove
- `=` -> set exactly (replacing existing permissions)

```
# add execute for user
chmod u+x file.sh
# remove write for group
chmod g-w file.txt
# set read only for others
chmod o=r file.log
```

Numeric (Octal) Mode

Each permission is defined by a number:

`r=4`, `w=2`, `x=1`

Add them up to combine permissions:

`7=rwx`, `6=rw-`, `5=r-x`, `4=r--`, `3=-wx`, `2=-w-`, `1=--x`, `0=---`

```
# rwx for user, r-x for group, r-x for others
chmod 755 script.sh
# rw- for user, r-- for group & others
chmod 644 file.txt
```

Think before use! Do not blindly use `chmod 777`!

Special Permissions

- Setuid: run program as file owner instead of current user
  - Example: `/usr/bin/passwd` (runs as root)
  - Check yourself: `ls -l /usr/bin/passwd`
  - Risky -> use carefully!
- Setgid: run program with effective GID of the file's group
  - On directories: new files inherit group
- Sticky Bit: users can only delete or rename files if they own the file or directory
  - Common on `/tmp`

```
chmod 4755 program   # setuid
chmod 2775 shared/   # setgid on directory
chmod 1777 /tmp      # sticky bit
```

Special characters instead of `x`:

- Lowercase: Execute granted
- Uppercase: Execute not granted

Permission	Meaning
`-rwsr-xr-x`	User `s`: Setuid
`-rwxr-sr-x`	Group `s`: Setgid
`drwxr-sr-x`	Setgid directory
`drwxrwxrwt`	Other `t`: Sticky bit

Managing File Permissions & Ownership

Command	Description	Useful Flags / Examples
`ls`	Show permissions and ownership	`ls -l`, `ls -ln`
`stat`	Show file/directory details	`stat -c '%a %n' file.txt`
`chmod`	Change file permissions	`chmod 644 file.txt`, `chmod +x script.sh`, `chmod -R 700 dir`
`chown`	Change owner/group	`chown user:group file.txt`, `chown user -R directory`
`umask`	Show/set default permission mask	`umask 0077` (default `0022`)

`umask` explained:

```
           New File          New Directory
Default:   666  rw-rw-rw-    777  rwxrwxrwx
Umask:     022  ----w--w-    022  ----w--w-
           --------------    -----------
Result:    644  rw-r--r--    755  rwxr-xr-x
```

Q: What does `umask 0077` do?

Newly created files can only be read and written by the owner.

> umask only affects the current session. If you want to keep a custom umask, add it to your ~/.bashrc

Exercise

Create a file and experiment

- Show and change your `umask`
  - Create new files (e.g. `touch myfile`) and compare permissions for different `umask`s
- Change file ownership with `chown foo:bar myfile`
  - Check which permissions are still available to your user

Use sticky directory

- The `/tmp` folder is created as sticky directory
  - `sudo mkdir /tmp`
  - `chmod 1777 /tmp`
- Try adding/removing files/directories as different users. Who can add/remove files?

Thank you!

Thank you for your attention!

Don't forget the feedback in Moodle please!