CSS 262: Linux Administration

Users, Groups & Permissions

Lecture 2: Access Control Fundamentals

📋 Today's Agenda

Part 1: Users & Groups

- Understanding multi-user systems

- User types and roles

- User management commands

- Group concepts

- System files (`/etc/passwd` , `/etc/shadow`)

Part 2: File Permissions

- Permission model (rwx)

- Reading permission strings

- chmod, chown, chgrp

- Special permissions (SUID, SGID, sticky bit)

- Access Control Lists (ACLs)

🎯 **Learning Objective:** Master Linux access control to secure systems and manage multi-user environments.

🔄 Quick Recap: Week 1

What We Covered

- The shell and command-line interface

- Basic navigation commands

- File system hierarchy

- Pipes and redirection

- Environment variables

Key Takeaway

The shell is your primary interface for system administration.

✅ **Assumption:** You now have a working Linux VM and can navigate the file system.

Part 1: Users & Groups

Understanding multi-user systems

Why Multi-User Systems?

Linux is Designed for Multiple Users

Linux inherited the multi-user concept from Unix (1970s) - multiple people sharing expensive mainframe computers.

Benefits

- **Isolation:** Users can't interfere with each other
- **Security:** Limit access to sensitive data
- **Resource Control:** Manage CPU, memory, disk usage
- **Accountability:** Track who did what
- **Flexibility:** Different roles and permissions

Use Cases

- Shared servers (web, database, etc.)
- Development environments
- University/corporate systems
- Cloud infrastructure
- Your VM (even with one human user!)

User Types in Linux

1. Root User (Superuser)

- **UID: 0** | Username: `root` | Home: `/root`
- Unlimited privileges - can do anything
- Dangerous - one wrong command can destroy system
- **Avoid direct login** - use `sudo` instead

2. Regular Users

- **UID: 1000+** | Home: `/home/username`
- Created by administrators
- Limited to their own files
- Can use `sudo` if granted

3. System Users

- **UID: 1-999** | Examples: `www-data`, `mysql`, `sshd`
- Run services and daemons
- No login shell (usually)

User Identifiers (UID)

Every User Has a Unique Number

```
1   id         # uid=1000(john) gid=1000(john) groups=1000(john),27(sudo)
2   id root    # uid=0(root) gid=0(root) groups=0(root)
```

UID Ranges

- **0** - Root user
- **1-999** - System users
- **1000+** - Regular users

Why UIDs Matter

- Permissions based on UID, not username
- File ownership stored as UID
- Process ownership tracked by UID
- Usernames are just friendly labels

Understanding Groups

Groups Enable Shared Access

A **group** is a collection of users who need similar permissions.

```
1    groups                   # john adm cdrom sudo dip plugdev
2    cat /etc/group | head -3 # root:x:0:  daemon:x:1:  bin:x:2:
```

Group Types

- **Primary group:** Every user has one
- **Supplementary:** Additional memberships
- **System groups:** For services

Common Groups

- `sudo` - Can use sudo command
- `wheel` - Admin group (some distros)
- `docker` - Can run Docker
- `www-data` - Web server files

The /etc/passwd File

The User Database (world-readable)

```
1   cat /etc/passwd | grep john
2   # john:x:1000:1000:John Doe,,,:/home/john:/bin/bash
```

7 Fields (colon-separated)

Field	Example	Description
1. Username	john	Login name
2. Password	x	Now in /etc/shadow
3. UID	1000	User ID number
4. GID	1000	Primary group ID
5. GECOS	John Doe	Full name
6. Home	/home/john	Home directory
7. Shell	/bin/bash	Login shell

The /etc/shadow File

Secure Password Storage (only root can read)

```
1   sudo cat /etc/shadow | grep john
2   # john:$6$rounds=656000$xyz ... :19345:0:99999:7:::
```

Field	Description
1. Username	Login name
2. Password	Encrypted with SHA-512 (`$6$`)
3. Last Changed	Days since 1970-01-01
4. Min Days	Min days between password changes
5. Max Days	Max days before must change
6. Warn Days	Days before expiry to warn
7-9.	Inactivity, expiry, reserved

Creating Users

useradd Command

```
1    # Create with home directory and shell
2    sudo useradd -m -s /bin/bash john
3
4    # Create with specific UID and groups
5    sudo useradd -m -u 1500 -G sudo,docker john
6
7    # Set password
8    sudo passwd john
```

Common Options

- `-m` - Create home directory

- `-s /bin/bash` - Set login shell

- `-G groups` - Add to supplementary groups

- `-u UID` - Specify UID

- `-c "Name"` - Set full name

- `-d /path` - Custom home directory

Modern User Creation: adduser

Interactive Wrapper (Debian/Ubuntu)

```
1    sudo adduser jane    # Interactive, prompts for details
```

useradd (Low-level)

- Manual configuration needed

- More control

- Works on all distros

```
1    sudo useradd -m -s /bin/bash jane
2    sudo passwd jane
```

adduser (High-level)

- Interactive and friendly

- Auto-creates home directory

- Debian/Ubuntu specific

```
1    sudo adduser jane
2    # That's it!
```

Managing Users

Modify Existing Users

```
 1    # Change user's shell
 2    sudo usermod -s /bin/zsh john
 3
 4    # Add user to group
 5    sudo usermod -aG sudo john      # -a = append, don't replace
 6
 7    # Change username
 8    sudo usermod -l newname oldname
 9
10    # Lock user account
11    sudo usermod -L john
12
13    # Unlock user account
14    sudo usermod -U john
```

Delete Users

```
 1    # Delete user (keep home directory)
 2    sudo userdel john
 3
```

Managing Groups

Group Management Commands

```
1    # Create a group
2    sudo groupadd developers
3
4    # Create with specific GID
5    sudo groupadd -g 5000 developers
6
7    # Add user to group
8    sudo usermod -aG developers john
9
10   # Or use gpasswd
11   sudo gpasswd -a john developers
12
13   # Remove user from group
14   sudo gpasswd -d john developers
15
16   # Delete a group
17   sudo groupdel developers
18
19   # View group members
20   getent group developers
```

sudo: Superuser Do

Run Commands as Root

```
1    sudo apt update              # Run as root
2    sudo -u www-data ls /var/www # Run as another user
3    sudo visudo                  # Edit /etc/sudoers safely
```

Why sudo?

- **Accountability:** Logs who ran what

- **Limited exposure:** Specific commands

- **No root password:** Don't share

- **Temporary:** Expires after 15 min

Configuration

```
1    # /etc/sudoers
2    john ALL=(ALL:ALL) ALL
3    %sudo ALL=(ALL:ALL) ALL
```

- `john` can run anything - `%sudo` group can run anything

su: Switch User

Switch to Another User

```
1    # Switch to another user
2    su - john      # Login shell (loads environment)
3    su john        # Non-login shell (keeps current env)
4
5    # Switch to root (requires root password)
6    su -
7    su - root
8
9    # Run single command as another user
10   su -c "whoami" john
```

su vs sudo

Feature	su	sudo
Password	Target user's password	Your password
Logging	Minimal	Detailed audit trail
Privilege	Becomes that user	Runs as that user
Best Practice	❌ Avoid for root	✅ Preferred method

Part 2: File Permissions

🔒
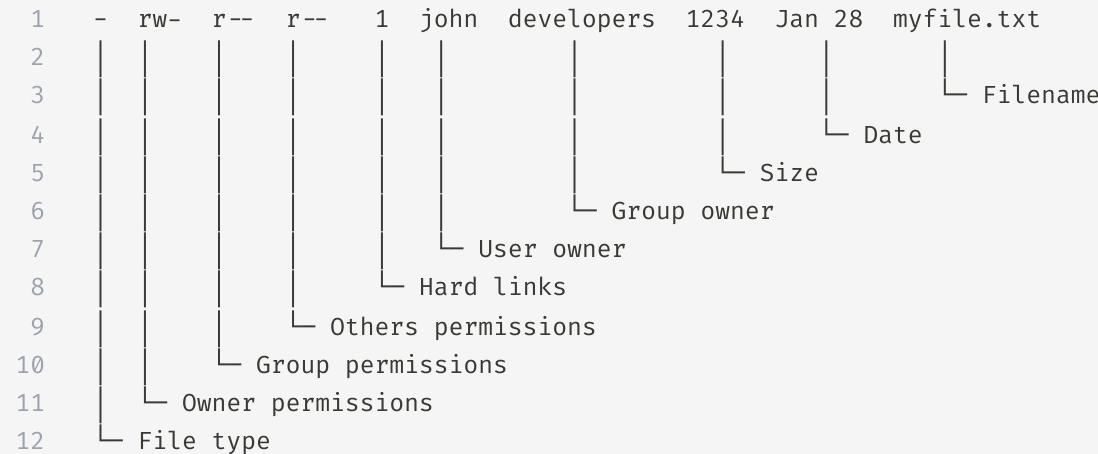
Controlling access to files and directories

Linux Permission Model

Every File Has Three Permission Sets

```
ls -l myfile.txt
-rw-r--r-- 1 john developers 1234 Jan 28 10:00 myfile.txt
```

Permission Breakdown

```
-   rw-   r--   r--    1   john   developers   1234   Jan 28   myfile.txt
|    |     |     |     |    |         |          |       |          |
|    |     |     |     |    |         |          |       |          └─ Filename
|    |     |     |     |    |         |          |       └─ Date
|    |     |     |     |    |         |          └─ Size
|    |     |     |     |    |         └─ Group owner
|    |     |     |     |    └─ User owner
|    |     |     |     └─ Hard links
|    |     |     └─ Others permissions
|    |     └─ Group permissions
|    └─ Owner permissions
└─ File type
```

File Types

First Character Indicates Type

Symbol	Type	Example
-	Regular file	`-rw-r--r--`
d	Directory	`drwxr-xr-x`
l	Symbolic link	`lrwxrwxrwx`
c	Character device	`crw-rw——` (keyboard)
b	Block device	`brw-rw——` (disk)
s	Socket	`srwxrwxrwx`
p	Named pipe	`prw-r--r--`

Examples

```
1   ls -la /dev | head -4
2   # drwxr-xr-x  20 root root   3940 Jan 28 10:00 .
3   # crw-rw-rw-   1 root tty  5,    0 Jan 28 14:45 tty
4   # brw-rw——    1 root disk 8,    0 Jan 28 10:00 sda
```

Permission Bits

For Files

- **r (read)** - View contents

    - `cat`, `less`, `grep`

- **w (write)** - Modify contents

    - `vim`, `echo >`, `rm`

- **x (execute)** - Run as program

    - `./script.sh`

```
1   -rw-r--r--   document.txt
2   -rwxr-xr-x   script.sh
3   -rw-------   secret.key
```

For Directories

- **r (read)** - List contents

    - `ls directory/`

- **w (write)** - Create/delete files

    - `touch`, `rm`, `mkdir`

- **x (execute)** - Enter directory

    - `cd directory/`

```
1   drwxr-xr-x   public/
2   drwx------   private/
3   drwxrwxrwx   tmp/
```

⚠️ **Important:** Need execute (x) on directory to access files!

Reading Permissions

Practice Exercise

```
1    -rwxr-xr--  1  john  developers  4096  Jan 28  script.sh
```

Owner (john)

- **r** - Can read
- **w** - Can write
- **x** - Can execute

Result: Full access

Group (developers)

- **r** - Can read
- **-** - Cannot write
- **x** - Can execute

Result: Read & execute only

Others (everyone else)

- **r** - Can read
- **-** - Cannot write
- **-** - Cannot execute

Result: Read only

chmod: Change Mode

Two Notations: Symbolic and Octal

Symbolic (Human-friendly)

```
 1    # Add execute for owner
 2    chmod u+x script.sh
 3
 4    # Remove write for group
 5    chmod g-w file.txt
 6
 7    # Set read-only for others
 8    chmod o=r file.txt
 9
10    # Add execute for all
11    chmod a+x script.sh
12
13    # Multiple changes
14    chmod u+x,g+x,o-r file.txt
```

Symbols: `u` =user, `g` =group, `o` =others, `a` =all

Octal (Numeric)

```
 1    # Read permissions as binary:
 2    # r w x
 3    # 4 2 1
 4
 5    # Examples:
 6    chmod 755 script.sh   # rwxr-xr-x
 7    chmod 644 file.txt    # rw-r--r--
 8    chmod 600 secret.key  # rw————
 9    chmod 777 public.sh   # rwxrwxrwx (dangerous!)
```

Common Values:

- `755` - Standard executable

- `644` - Standard file

- `600` - Private file

- `700` - Private directory

chmod Examples

```
1   # Make script executable
2   chmod +x script.sh                  # Same as chmod a+x
3   chmod 755 script.sh
4
5   # Protect sensitive file
6   chmod 600 ~/.ssh/id_rsa             # Only owner can read/write
7
8   # Public readable file
9   chmod 644 document.txt              # Owner: rw, others: r
10
11  # Shared directory
12  chmod 770 /shared/project           # Owner+Group: full, Others: none
13
14  # Recursive permissions
15  chmod -R 755 /var/www/html          # Apply to all files/dirs
16
17  # Remove all permissions for others
18  chmod o-rwx file.txt
19
20  # Set specific permissions
21  chmod u=rwx,g=rx,o=r file.txt       # rwxr-xr--
```

Octal Permissions Cheat Sheet

Binary to Octal Conversion

Binary	Octal	Perms	Description
000	0	`---`	No access
001	1	`--x`	Execute only
010	2	`-w-`	Write only
011	3	`-wx`	Write + Execute
100	4	`r--`	Read only
101	5	`r-x`	Read + Execute
110	6	`rw-`	Read + Write
111	7	`rwx`	Full access

Common Patterns

`644` → `rw-r--r--` (file) | `755` → `rwxr-xr-x` (executable) | `700` → `rwx------` (private) | `777` → ⚠️ dangerous!

chown: Change Owner

Change File Ownership

```
1    # Change owner
2    sudo chown john file.txt
3
4    # Change owner and group
5    sudo chown john:developers file.txt
6
7    # Change only group (or use chgrp)
8    sudo chown :developers file.txt
9
10   # Recursive ownership change
11   sudo chown -R john:developers /var/www/mysite
12
13   # Copy ownership from another file
14   sudo chown --reference=file1.txt file2.txt
```

Why Root Required?

> 💡 **Tip:** When deploying web apps, set ownership to `www-data:www-data` for web server access.

chgrp: Change Group

Change File Group

```
1    # Change group ownership
2    chgrp developers file.txt
3
4    # Recursive group change
5    chgrp -R developers /shared/project
6
7    # Using chown (alternative)
8    chown :developers file.txt
```

Practical Example: Shared Project

```
1    # Create shared directory for team
2    sudo mkdir /shared/project
3    sudo chgrp developers /shared/project
4    sudo chmod 770 /shared/project
5
6    # Now all users in 'developers' group can collaborate
7    # Owner: rwx
8    # Group: rwx (all developers)
9    # Others: --- (no access)
```

Default Permissions: umask

umask Controls Default Permissions

```
1    umask        # Output: 0022
2    umask -S     # Output: u=rwx,g=rx,o=rx
```

How umask Works

umask **subtracts** from maximum permissions:

- Files: `666` - umask

- Directories: `777` - umask

umask	New Files	New Directories
022	`644` (rw-r–r–)	`755` (rwxr-xr-x)
077	`600` (rw-------)	`700` (rwx------)
002	`664` (rw-rw-r–)	`775` (rwxrwxr-x)

Special Permissions (Part 1)

SUID (Set User ID) - Bit 4000

Runs with the **owner's** privileges, not the user who runs it.

```
1   # Set SUID
2   chmod u+s /usr/bin/passwd
3   chmod 4755 /usr/bin/passwd
4
5   # Check SUID
6   ls -l /usr/bin/passwd
7   # -rwsr-xr-x 1 root root 68208 passwd
8   #    ^  (s = SUID bit)
```

Real Example: passwd Command

`passwd` needs to modify `/etc/shadow` (root-only), but regular users need to change passwords. SUID allows it to run as root.

⚠️ **Security Risk:** SUID scripts are dangerous!

Special Permissions (Part 2)

SGID (Set Group ID) - Bit 2000

On files: Runs with the group privileges

On directories: New files inherit the directory's group

```
1    # Set SGID on directory
2    chmod g+s /shared/project
3    chmod 2770 /shared/project
4
5    ls -l /shared
6    # drwxrws--- 2 john developers 4096 project
7    #       ^  (s = SGID bit)
```

Practical Use: Shared Directories

Without SGID: files get user's primary group

With SGID: files get directory's group (developers)

Result: All team members can access new files!

Special Permissions (Part 3)

Sticky Bit - Bit 1000

On directories: Only owner can delete their files.

```
1    chmod +t /tmp
2    chmod 1777 /tmp
3
4    ls -ld /tmp
5    # drwxrwxrwt 20 root root 4096 /tmp
6    #          ^  (t = sticky bit)
```

Real Example: /tmp Directory

Everyone can create files in `/tmp` (777 permissions), but you can't delete someone else's files (sticky bit). Prevents malicious deletion.

SUID: 4000 (u+s)   **SGID:** 2000 (g+s)   **Sticky:** 1000 (+t)

Access Control Lists (ACLs)

Beyond Traditional Permissions

ACLs allow fine-grained permissions for multiple users/groups.

```
1    # Grant read access to specific user
2    setfacl -m u:alice:r file.txt
3
4    # Grant read+write to specific group
5    setfacl -m g:managers:rw file.txt
6
7    # Remove ACL / View ACLs
8    setfacl -x u:alice file.txt
9    getfacl file.txt
10
11   # View output
12   # user::rw-
13   # user:alice:r--
14   # group::r--
15   # group:managers:rw-
```

ACL Examples

Scenario: Shared File with Exceptions

```
1   # Give alice (not in developers) read access
2   setfacl -m u:alice:r document.txt
3
4   # Check result
5   ls -l document.txt
6   # -rw-r--r--+ 1 john developers 1234 document.txt
7   #           ^  '+' indicates ACL is set
```

Default ACLs for Directories

```
1   # Set default ACL (applies to new files)
2   setfacl -d -m g:developers:rwx /shared/project
3
4   # All new files inherit developers:rwx
```

Permission Troubleshooting

Common Issues

Problem	Cause	Solution
"Permission denied" reading	No read permission	`chmod +r file`
Can't enter directory	No execute permission	`chmod +x dir/`
Can't delete file	No write on directory	`chmod +w dir/`
Script won't execute	No execute bit	`chmod +x script.sh`
"Operation not permitted"	Not owner/not root	Use `sudo`

Debug Commands

```
1   id                    # Check your identity
2   ls -l file.txt        # Check file permissions
3   ls -ld directory/     # Check directory permissions
4   sudo cat /etc/shadow  # Test with sudo
```

Security Best Practices

DO ✅

- Use `chmod 600` for private keys
- Use `chmod 644` for regular files
- Use `chmod 755` for executables
- Use `sudo` instead of root login
- Set restrictive umask (022 or 027)
- Use groups for shared access
- Audit SUID/SGID files regularly
- Remove world-writable permissions

DON'T ❌

- `chmod 777` (world-writable!)
- Run services as root
- Share root password
- Give SUID to scripts
- Leave default passwords
- Ignore permission errors
- Grant unnecessary sudo access
- Forget to revoke old accounts

🛡️ **Security Principle:** Principle of Least Privilege - grant only the minimum permissions needed.

Finding Files by Permission

Security Audit Commands

```
1    # Find all SUID files (security risk)
2    find / -perm -4000 -type f 2>/dev/null
3
4    # Find all SGID files
5    find / -perm -2000 -type f 2>/dev/null
6
7    # Find world-writable files
8    find / -perm -002 -type f 2>/dev/null
9
10   # Find files owned by user
11   find / -user john 2>/dev/null
12
13   # Find orphaned files (no owner)
14   find / -nouser 2>/dev/null
15
16   # Find recent modifications
17   find / -mtime -7 -type f 2>/dev/null
```

Lab 2 Preview: User Management

What You'll Do

1. 👤 Create multiple user accounts

2. 👥 Create groups and manage membership

3. 🔒 Set appropriate file permissions

4. 🔑 Configure sudo access

5. 🗂 Set up shared directory with SGID

6. 🔍 Audit permission issues

Deliverables

- Screenshot of user creation

- Output of `id`, `groups` commands

- Properly configured shared directory

- Fixed permission problems

- Working sudo configuration

Time Estimate: 2 hours

Practical Scenarios

Scenario 1: Web Application

```
1   sudo mkdir -p /var/www/myapp
2   sudo chown www-data:www-data /var/www/myapp
3   sudo chmod 755 /var/www/myapp
```

Scenario 2: Shared Development

```
1   sudo groupadd devteam
2   sudo mkdir /projects/webapp
3   sudo chown :devteam /projects/webapp
4   sudo chmod 2775 /projects/webapp  # SGID + rwxrwxr-x
```

Scenario 3: Secure Backup

```
1   chmod 600 backup.tar.gz
2   chown backup-user:backup-group backup.tar.gz
```

Quick Reference Card

Command	Purpose	Example
`useradd`	Create user	`sudo useradd -m john`
`passwd`	Set password	`sudo passwd john`
`usermod`	Modify user	`sudo usermod -aG sudo john`
`userdel`	Delete user	`sudo userdel -r john`
`groupadd`	Create group	`sudo groupadd devs`
`chmod`	Change permissions	`chmod 755 file.sh`
`chown`	Change owner	`sudo chown john:devs file`
`chgrp`	Change group	`chgrp devs file`
`umask`	Default permissions	`umask 022`
`id`	Show user info	`id john`
`groups`	Show groups	`groups john`
`getfacl`	Show ACLs	`getfacl file.txt`
`setfacl`	Set ACLs	`setfacl -m u:alice:r file`

Important Files Summary

File	Purpose	Permissions
`/etc/passwd`	User account info	`-rw-r--r--` (644)
`/etc/shadow`	Encrypted passwords	`-rw-r-----` (640)
`/etc/group`	Group definitions	`-rw-r--r--` (644)
`/etc/gshadow`	Group passwords	`-rw-r-----` (640)
`/etc/sudoers`	Sudo configuration	`-r--r-----` (440)
`/home/*`	User home directories	`drwx------` (700)
`~/.ssh/`	SSH keys	`drwx------` (700)
`~/.ssh/id_rsa`	Private SSH key	`-rw-------` (600)

⚠️ **Never manually edit /etc/passwd or /etc/shadow!** Use proper commands.

Common Mistakes to Avoid

Permission Mistakes

- `chmod 777` on everything
- Forgetting execute bit on directories
- Wrong ownership on web files
- SUID on shell scripts
- World-readable private keys
- Overly permissive sudo access

User Management Mistakes

- Sharing root password
- Not using `sudo`
- Forgetting to delete old users
- Wrong primary group
- No password policy
- Not locking unused accounts
- Hardcoding UIDs in scripts

Week 2 Action Items

✅ Before Next Lecture

1. Read **Chapter 6** of "The Linux Command Line" (Permissions)

2. Practice permission commands on your VM

3. Create test users and groups

4. Experiment with SUID, SGID, sticky bit

✅ For Lab This Week

1. Complete **Lab 2: User Management**

2. Set up multi-user environment

3. Configure shared directories

4. Test permission scenarios

5. Debug permission issues

📝 Practice Exercises

- Find all SUID files on your system

- Create a shared project directory with proper permissions

Questions?

?

Understanding permissions is crucial for system security!

Next Week: Process Management & Systemd ⚙️

Thank You!

🔒🔑

Remember: Proper permissions = Secure systems

CSS 262 - Linux Administration & *nix Systems for Cybersecurity