Users and Groups in Linux

Linux is a multi-user operating system, which means it supports multiple users accessing the system simultaneously. Understanding users and groups is fundamental for managing permissions, securing resources, and facilitating collaboration.

Table of Contents

Introduction to Users and Groups

In Linux:

- A **user** is an entity that can log in and perform operations.
- A **group** is a collection of users, used to simplify permission management.

Each user has:

1. **UID**: A unique identifier.
2. **Home Directory**: A directory for personal files.
3. **Shell**: The default command-line interpreter.

Each group has:

- A **GID** (Group ID), which is a unique identifier.

User Management

Adding Users

Create a new user with the `useradd` command.

Example:

```
sudo useradd -m -s /bin/bash john
```

- `-m`: Creates a home directory (`/home/john`).
- `-s`: Specifies the shell (`/bin/bash`).

Set a password for the user:

```
sudo passwd john
```

Verify:

```
id john
```

Output:

```
uid=1001(john) gid=1001(john) groups=1001(john)
```

Deleting Users

Remove a user with `userdel`.

Example:

```
sudo userdel -r john
```

- `-r`: Deletes the user's home directory and mail spool.

Modifying Users

Modify user information with `usermod`.

Examples:

1. Change a user's shell:

   ```
   sudo usermod -s /bin/zsh john
   ```

2. Add a user to a supplementary group:

   ```
   sudo usermod -aG sudo john
   ```

   - `-aG`: Appends the user to the specified group without removing existing group memberships.

Group Management

Adding Groups

Create a new group with `groupadd`.

Example:

```
sudo groupadd developers
```

Deleting Groups

Remove a group with `groupdel`.

Example:

```
sudo groupdel developers
```

Modifying Groups

Change group membership with `gpasswd`.

Example: Add a user to a group:

```
sudo gpasswd -a john developers
```

Remove a user from a group:

```
sudo gpasswd -d john developers
```

`useradd` vs `adduser`

`useradd`

- A low-level command for adding users.
- Requires explicit options for additional setup (e.g., home directory, shell).
- Syntax:

  ```
  sudo useradd [options] username
  ```

Example: Create a User with a Home Directory

```
sudo useradd -m -s /bin/bash alice
```

`adduser`

- A higher-level, interactive command (script) for adding users.
- Automatically prompts for details like full name, password, and groups.
- Syntax:

  ```
  sudo adduser username
  ```

Example: Create a User Interactively

```
sudo adduser bob
```

Creating Multiple Users and Groups

Create Multiple Users with `useradd`

Example:

```
sudo useradd -m -s /bin/bash user1
sudo useradd -m -s /bin/bash user2
sudo useradd -m -s /bin/bash user3
```

Create Groups

Example:

```
sudo groupadd groupA
sudo groupadd groupB
```

Demo: Assigning Users to Groups

Scenario Setup

1. **Users**: `user1`, `user2`, and `user3`.
2. **Groups**:
   - `groupA`: `user1`, `user2`
   - `groupB`: `user2`, `user3`

Commands

Step 1: Add Users

```
sudo useradd -m -s /bin/bash user1
sudo useradd -m -s /bin/bash user2
sudo useradd -m -s /bin/bash user3
```

Step 2: Create Groups

```
sudo groupadd groupA
sudo groupadd groupB
```

Step 3: Assign Users to Groups

- Using `gpasswd`:

  ```
  sudo gpasswd -a user1 groupA
  sudo gpasswd -a user2 groupA
  sudo gpasswd -a user2 groupB
  sudo gpasswd -a user3 groupB
  ```

Verify Group Membership

```
groups user1
groups user2
groups user3
```

File and Group Ownership Commands

Explanation of `chown`

The `chown` command changes user and/or group ownership of a file or directory.

Syntax:

```
sudo chown [USER]:[GROUP] FILE
```

Examples:

1. Change user ownership only:

   ```
   sudo chown alice file.txt
   ```

2. Change both user and group ownership:

   ```
   sudo chown alice:developers file.txt
   ```

3. Recursively change ownership:

```
sudo chown -R alice:developers /shared
```

Explanation of `chgrp`

The `chgrp` command changes only the group ownership of a file or directory.

Syntax:

```
sudo chgrp [GROUP] FILE
```

Examples:

1. Change group ownership:

   ```
   sudo chgrp developers file.txt
   ```

2. Change group ownership recursively:

   ```
   sudo chgrp -R developers /shared
   ```

Demo: Changing Group Ownership (`chgrp`)

Scenario:

- File: `project.txt`
- Initial group ownership: `groupA`
- Desired group ownership: `groupB`

Commands:

```
sudo touch project.txt
sudo chown :groupA project.txt
sudo chgrp groupB project.txt
```

Verify Changes:

```
ls -l project.txt
```

Demo: Proof of Concept with Users and Groups

Note: We start as the primary user `mahmoud`, who will create and manage all other users and groups.

Goal

1. Create users and groups under `mahmoud`:
   - `user1` in `groupA`
   - `user2` in both `groupA` and `groupB`

- `user3` in `groupB`

2. `user1` creates a file accessible only to `groupA`.
3. Verify that `user3` (in `groupB` only) cannot access the file.
4. Add a new user (`user4`) to `groupA` and verify that `user4` can access the file.
5. Change the file's group ownership to `groupB` and verify that now `user3` can access it, but `user4` (in `groupA` only) cannot.

Steps

1. Create Groups and Users

```
# As mahmoud (with sudo privileges):

# Create groups
sudo groupadd groupA
sudo groupadd groupB

# Create users with primary groups
sudo useradd -m -s /bin/bash -g groupA user1
sudo useradd -m -s /bin/bash -g groupA user2
sudo useradd -m -s /bin/bash -g groupB user3

# Add user2 to groupB as a supplementary group
sudo usermod -aG groupB user2

# Set passwords (will be prompted)
sudo passwd user1
sudo passwd user2
sudo passwd user3
```

After this setup:

- `user1`: primary group `groupA`
- `user2`: primary group `groupA`, supplementary `groupB`
- `user3`: primary group `groupB`

2. `user1` Creates a File

```
su - user1
touch project.txt
chown user1:groupA project.txt
chmod 660 project.txt
# Permissions: rw-rw---- (Owner: user1, Group: groupA, Others: none)
ls -l project.txt
```

Expected:

```
-rw-rw---- 1 user1 groupA 0 [timestamp] project.txt
```

Exit back to `mahmoud`:

```
exit
```

3. `user3` Tries to Access the File

```
su - user3
cat /home/user1/project.txt
```

Expected Result:

```
cat: /home/user1/project.txt: Permission denied
```

Exit back:

```
exit
```

4. Add `user4` to `groupA` and Test Access

```
sudo useradd -m -s /bin/bash -g groupA user4
sudo passwd user4
```

Verify `user4` is in `groupA`:

```
id user4
```

Switch to `user4`:

```
su - user4
cat /home/user1/project.txt
# Should succeed because user4 is in groupA
exit
```

5. Change Group Ownership to `groupB`

```
sudo chown user1:groupB /home/user1/project.txt
ls -l /home/user1/project.txt
```

Expected:

```
-rw-rw---- 1 user1 groupB 0 [timestamp] project.txt
```

6. Verify Access After Changing Group Ownership

- `user3` (groupB) now tries again:

  ```
  su - user3
  cat /home/user1/project.txt
  # Should now succeed
  exit
  ```

- `user4` (groupA only) tries again:

```
su - user4
cat /home/user1/project.txt
# Should now fail, since the file belongs to groupB
exit
```

Summary of Results

- Initially, `project.txt` was accessible only by `groupA`.
  - `user3` (groupB) couldn't access it.
  - `user4` (groupA) could.
- After changing group ownership to `groupB`, the situation reversed:
  - `user3` (groupB) could now access the file.
  - `user4` (groupA) could no longer access it.

Additional Tips and Tricks

Primary Groups vs. Supplementary Groups

- **Primary Group**:
  Each user has exactly one primary group. This group is typically the one specified when the user was created. Files created by the user usually default to being owned by the user's primary group. The primary group is stored in `/etc/passwd` and is often a private group unique to the user, though it can also be a shared group.
- **Supplementary (Secondary) Groups**:
  A user can belong to multiple supplementary groups. These are additional groups that grant the user extra permissions and access beyond what their primary group provides. Supplementary groups are stored in `/etc/group`. Being part of multiple supplementary groups allows a single user to easily share files and resources with different sets of colleagues or projects, without changing their primary group.

Assigning Users to Groups

There are multiple ways to assign users to groups:

1. **During User Creation (`useradd`)**:
   - `-g`: Sets the user's primary group.
     Example:

     ```
     sudo useradd -m -s /bin/bash -g groupA newuser
     ```

     This makes `groupA` the primary group of `newuser`.

- `-G`: Sets the user's supplementary (secondary) groups.
  Example:

  ```
  sudo useradd -m -s /bin/bash -g groupA -G groupB,groupC newuser
  ```

  This creates `newuser` with `groupA` as the primary group and adds `groupB` and `groupC` as supplementary groups.

2. **After User Creation**:
   - Using `usermod`:
     - `-g` with `usermod` changes the primary group.
     - `-G` with `usermod` sets the user's supplementary groups, **overwriting** any existing supplementary groups. To avoid overwriting, use `-aG` to append groups:

       ```
       sudo usermod -g groupA user1          # Change primary group
       sudo usermod -G groupB,groupC user1   # Sets supplementary groups to ONLY...
                                             # groupB and groupC
       sudo usermod -aG groupD user1         # Appends groupD to the current...
                                             # supplementary groups
       ```

   - Using `gpasswd -a`:

     ```
     sudo gpasswd -a user1 groupA
     ```

     This adds `user1` to `groupA` without removing other supplementary groups.

Key Differences:

- `-g` (lowercase) sets or changes the **primary** group of a user.
- `-G` (uppercase) sets or changes the **supplementary** groups of a user.
  When using `usermod -G`, it replaces the current supplementary group list.
  Use `-aG` to append instead of replace.