MANAGING LOCAL USERS AND GROUPS

Objectives

- Describe the purpose of users and groups on a Linux system.
- Switch to the superuser account to manage a Linux system, and grant other users superuser access using the sudo command.
- Create, modify, and delete locally defined user accounts.
- Create, modify, and delete locally defined group accounts.
- Set a password management policy for users, and manually lock and unlock user accounts.

DESCRIBING USER AND GROUP CONCEPTS

1. What is a USER?

A user account is used to provide security boundaries between different people and programs that can run commands.

Users have user names to identify them to human users and make them easier to work with. Internally, the system distinguishes user accounts by the unique identification number assigned to them, the user ID or UID. If a user account is used by humans, it will generally be assigned a secret password that the user will use to prove that they are the actual authorized user when logging in.

User accounts are fundamental to system security. Every process (running program) on the system runs as a particular user. Every file has a particular user as its owner. File ownership helps the system enforce access control for users of the files. The user associated with a running process determines the files and directories accessible to that process.

There are three main types of user account: the superuser, system users, and regular users.

- The ***superuser*** account is for administration of the system. The name of the superuser is root and the account has UID 0. The superuser has full access to the system.
- The system has ***system user*** accounts which are used by processes that provide supporting services. These processes, or daemons, usually do not need to run as the superuser. They are assigned non-privileged accounts that allow them to secure their files and other resources from each other and from regular users on the system. Users do not interactively log in using a system user account.

- Most users have ***regular user*** accounts which they use for their day-to-day work. Like system users, regular users have limited access to the system.

You can use the **id** command to show information about the currently logged-in user.

```
[lokesh@localhost ~]$ id
uid=1000(user01) gid=1000(user01) groups=1000(user01)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

To view basic information about another user, pass the username to the id command as an argument.

```
[lokesh@localhost]$ id user02
uid=1002(user02) gid=1001(user02) groups=1001(user02)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

To view the owner of a file use the ls **-l** command. To view the owner of a directory use the ls -**ld** command. In the following output, the third column shows the username.

```
[lokesh@localhost ~]$ ls -l file1
-rw-rw-r--. 1 user01 user01 0 Feb 5 11:10 file1
[lokesh@localhost]$ ls -ld dir1
drwxrwxr-x. 2 user01 user01 6 Feb 5 11:10 dir1
```

To view process information, use the **ps** command. The default is to show only processes in the current shell. Add the **a** option to view all processes with a terminal. To view the user associated with a process, include the **u** option.

In the following output, the first column shows the username.

```
[user01@host]$ ps -au
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
root 777 0.0 0.0 225752 1496 tty1 Ss+ 11:03 0:00 /sbin/agetty -o -
p -- \u --noclear tty1 linux
root 780 0.0 0.1 225392 2064 ttyS0 Ss+ 11:03 0:00 /sbin/agetty -o -
p -- \u --keep-baud 115200,38400,9600
user01 1207 0.0 0.2 234044 5104 pts/0 Ss 11:09 0:00 -bash
user01 1319 0.0 0.2 266904 3876 pts/0 R+ 11:33 0:00 ps au
```

The output of the preceding command displays users by name, but internally the operating system uses the UIDs to track users. The mapping of usernames to UIDs is defined in databases of account information. By default, systems use the **/etc/passwd** file to store information about local users.

Each line in the **/etc/passwd** file contains information about one user. It is divided up into seven colon-separated fields. Here is an example of a line from **/etc/passwd:**

```
❶user01:❷x:❸1000:❹1000:❺User One:❻/home/user01:❼/bin/bash
```

1) Username for this user (user01).
2) The user's password used to be stored here in encrypted format. That has been moved to the **/etc/shadow** file, which will be covered later. This field should always be x.
3) The UID number for this user account (1000).
4) The GID number for this user account's primary group (1000).
5) The real name for this user (User One).
6) The home directory for this user (**/home/user01**). This is the initial working directory when the shell starts and contains the user's data and configuration settings.
7) The default shell program for this user, which runs on login (**/bin/bash**). For a regular user, this is normally the program that provides the user's command-line prompt. A system user might use **/sbin/nologin** if interactive logins are not allowed for that user.

2. What is a GROUP?

A group is a collection of users that need to share access to files and other system resources. Groups can be used to grant access to files to a set of users instead of just a single user.

Like users, groups have group names to make them easier to work with. Internally, the system distinguishes groups by the unique identification number assigned to them, the group ID or GID.

The mapping of group names to GIDs is defined in databases of group account information. By default, systems use the **/etc/group** file to store information about local groups.

Each line in the **/etc/group** file contains information about one group. Each group entry is divided into four colon-separated fields. Here is an example of a line from **/etc/group**:

```
❶group01:❷x:❸10000:❹user01,user02,user03
```

1) Group name for this group (**group01**).
2) Obsolete group password field. This field should always be **x**.
3) The GID number for this group (**10000**).
4) A list of users who are members of this group as a supplementary group (**user01, user02, user03**).

Primary Groups and Supplementary Groups

Every user has exactly one primary group. For local users, this is the group listed by GID number in the **/etc/passwd** file. By default, this is the group that will own new files created by the user.

Normally, when you create a new regular user, a new group with the same name as that user is created. That group is used as the primary group for the new user, and that user is the only member of this User Private Group. It turns out that this helps make management of file permissions simpler, which will be discussed later in this course.

Users may also have supplementary groups. Membership in supplementary groups is determined by the **/etc/group** file. Users are granted access to files based on whether any of their groups have access. It doesn't matter if the group or groups that have access are primary or supplementary for the user.

For example, if the user **user01** has a primary group **user01** and supplementary groups wheel and **webadmin**, then that user can read files readable by any of those three groups.

The **id** command can also be used to find out about group membership for a user.

```
[user03@host ~]$ id
uid=1003(user03) gid=1003(user03) groups=1003(user03),10(wheel),10000(group01)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

In the preceding example, user03 has the group user03 as their primary group (gid). The groups item lists all groups for this user, and other than the primary group user03, the user has groups wheel and group01 as supplementary groups.

GAINING SUPERUSER ACCESS

1. The SuperUser?

Most operating systems have some sort of **superuser**, a user that has all power over the system. In Red Hat Enterprise Linux this is the root user. This user has the power to override normal privileges on the file system, and is used to manage and administer the system. To perform tasks such as installing or removing software and to manage system files and directories, users must escalate their privileges to the root user.

This unlimited privilege, however, comes with responsibility. The root user has unlimited power to damage the system: remove files and directories, remove user accounts, add back doors, and so on. If the root user's account is compromised, someone else would have administrative control of the system.

The root account on Linux is roughly equivalent to the local Administrator account on Microsoft Windows. In Linux, most system administrators log in to the system as an unprivileged user and use various tools to temporarily gain root privileges.

2. Switching Users

The **su** command allows users to switch to a different user account. If you run **su** from a regular user account, you will be prompted for the password of the account to which you want to switch. When **root** runs **su**, you do not need to enter the user's password.

```
[user01@host ~]$ su - user02

Password:

[user02@host ~]$
```

If you omit the user name, the **su** or **su** - command attempts to switch to root by default.

```
[user01@host ~]$ su -
Password:
[root@host ~]#
```

The command **su** starts a *non-login shell*, while the command **su -** (with the dash option) starts a login shell. The main distinction between the two commands is that **su -** sets up the shell environment as if it were a new login as that user, while **su** just starts a shell as that user, but uses the original user's environment settings.

In most cases, administrators should run **su -** to get a shell with the target user's normal environment settings.

3. Running Commands with SUDO

In some cases, the root user's account may not have a valid password at all for security reasons. In this case, users cannot log in to the system as root directly with a password, and **su** cannot be used to get an interactive shell. One tool that can be used to get root access in this case is **sudo**.

Unlike **su, sudo** normally requires users to enter their own password for authentication, not the password of the user account they are trying to access. That is, users who use **sudo** to run commands as root do not need to know the root password. Instead, they use their own passwords to authenticate access.

Additionally, **sudo** can be configured to allow specific users to run any command as some other user, or only some commands as that user.

For example, when **sudo** is configured to allow the user01 user to run the command **usermod** as root, user01 could run the following command to lock or unlock a user account:

```
[user01@host ~]$ sudo usermod -L user02
[sudo] password for user01:
[user01@host ~]$ su - user02
Password:
su: Authentication failure
[user01@host ~]$
```

If a user tries to run a command as another user, and the **sudo** configuration does not permit it, the command will be blocked, the attempt will be logged, and by default an email will be sent to the **root** user.

```
[user02@host ~]$ sudo tail /var/log/secure
[sudo] password for user02:
user02 is not in the sudoers file. This incident will be reported.
[user02@host ~]$
```

One additional benefit to using sudo is that all commands executed are logged by default to **/var/log/secure.**

```
[user01@host ~]$ sudo tail /var/log/secure
...output omitted...
Feb 6 20:45:46 host sudo[2577]: user01 : TTY=pts/0 ; PWD=/home/user01 ;
USER=root ; COMMAND=/sbin/usermod -L user02
...output omitted...
```

In Red Hat Enterprise Linux 8, all members of the wheel group can use **sudo** to run commands as any user, including root. The user is prompted for their own password.

Getting an Interactive Root Shell with Sudo

If there is a nonadministrative user account on the system that can use **sudo** to run the **su** command, you can run **sudo su -** from that account to get an interactive root user shell. This works because **sudo** will run **su -** as root, and root does not need to enter a password to use **su**.

Configuring Sudo

The main configuration file for **sudo** is **/etc/sudoers**. To avoid problems if multiple administrators try to edit it at the same time, it should only be edited with the special **visudo** command.

For example, the following line from the **/etc/sudoers** file enables **sudo** access for members of group wheel.

```
%wheel        ALL=(ALL)       ALL
```

In this line, **%wheel** is the user or group to whom the rule applies. A % specifies that this is a group, group wheel. The **ALL=(ALL)** specifies that on any host that might have this file, wheel can run any command. The final ALL specifies that wheel can run those commands as any user on the system.

By default, **/etc/sudoers** also includes the contents of any files in the **/etc/sudoers.d** directory as part of the configuration file. This allows an administrator to add **sudo** access for a user simply by putting an appropriate file in that directory.

To enable full **sudo** access for the user ***user01***, you could create **/etc/sudoers.d/**user01 with the following content:

```
user01          ALL=(ALL)       ALL
```

To enable full sudo access for the group group01, you could create **/etc/sudoers.d/group01** with the following content:

```
%group01        ALL=(ALL)       ALL
```

It is also possible to set up **sudo** to allow a user to run commands as another user without entering their password:

```
ansible         ALL=(ALL)       NOPASSWD:ALL
```

While there are obvious security risks to granting this level of access to a user or group, it is frequently used with cloud instances, virtual machines, and provisioning systems to help configure servers.

Tasks

1. Open an SSH session to localhost as pllab
2. Explore the shell environment of pllab
    a. View the current user and group information
    b. Display the current working directory
    c. Print the values of the **HOME** and **PATH** variables to determine the home directory and user executable's path, respectively.
3. Switch to **root** in a non-login shell and explore the new shell environment
    a. Run **sudo su**
    b. Run **id**
    c. Run **pwd**
    d. Print the values of the **HOME** and **PATH** variables to determine the home directory and user executable's path, respectively.
    e. Exit from the **root** user's shell to return to the **pllab** user's shell
4. Switch to **root** in login shell and explore the new shell environment
    a. Run **sudo su**
    b. Run **id**
    c. Run **pwd**
    d. Print the values of the **HOME** and **PATH** variables to determine the home directory and user executable's path, respectively.
    e. Exit from the **root** user's shell to return to the **pllab** user's shell
5. Create a new user **operator1**
6. Verify that the **operator1** user is configured to run any command as any user using **sudo**
7. Become **operator1** and view the contents on **/var/log/messages**. Copy **/etc/motd** to **/etc/motdOLD** and remove it. These operations require administrative rights and so use **sudo** to run these commands as the superuser. Do not switch to root using **sudo su** or **sudo su –**

MANAGING LOCAL USER ACCOUNTS

1. Managing Local Users

A number of command-line tools can be used to manage local user accounts.

Creating Users from the Command Line

- The **useradd *username*** command creates a new user named username. It sets up the user's home directory and account information, and creates a private group for the user named username. At this point the account does not have a valid password set, and the user cannot log in until a password is set.
- The **useradd --help** command displays the basic options that can be used to override the defaults. In most cases, the same options can be used with the **usermod** command to modify an existing user.
- Some defaults, such as the range of valid UID numbers and default password aging rules, are read from the **/etc/login.defs** file. Values in this file are only used when creating new users.

Modifying Existing Users from the Command Line

- The **usermod - - help** command displays the basic options that can be used to modify an account. Some options include:

USERMOD OPTIONS:	**USAGE**
-c, --comment COMMENT	Add the user's real name to the comment field.
-g, --gid GROUP	Specify the primary group for the user account
-G, --groups GROUPS	Specify a comma-separated list of supplementary groups for the user account

-a, --append	Used with the **–G** option to add the supplementary groups to the user's current set of group memberships instead of replacing the set of supplementary groups with a new set.
-d, --home HOME_DIR	Specify a particular home directory for the user account
-m, --move-home	Move the user's home directory to a new location. Must be used with **–d** option
-s, --shell SHELL	Specify a particular login shell for the user account
-L, --lock	Lock the user account
-U, --unlock	Unlock the user account

Deleting Users from the Command Line

- The **userdel *username*** command removes the details of username from **/etc/passwd,** but leaves the user's home directory intact.
- The **userdel -r *username*** command removes the details of username from **/etc/passwd** and also deletes the user's home directory.

Setting Passwords from the Command Line

- The p**asswd *username*** command sets the initial password or changes the existing password of username.
- The root user can set a password to any value. A message is displayed if the password does not meet the minimum recommended criteria, but is followed by a prompt to retype the new password and all tokens are updated successfully.

```
[root@host ~]# passwd user01
Changing password for user user01.
New password: redhat
BAD PASSWORD: The password fails the dictionary check - it is based on a
dictionary word
Retype new password: redhat
passwd: all authentication tokens updated successfully.
[root@host ~]#
```

- A regular user must choose a password at least eight characters long and is also not based on a dictionary word, the username, or the previous password.

UID Ranges

Specific UID numbers and ranges of numbers are used for specific purposes by Red Hat Enterprise Linux.

- UID 0 is always assigned to the superuser account, root.
- UID 1-200 is a range of "system users" assigned statically to system processes by Red Hat.
- UID 201-999 is a range of "system users" used by system processes that do not own files on the file system. They are typically assigned dynamically from the available pool when the software that needs them is installed. Programs run as these "unprivileged" system users in order to limit their access to only the resources they need to function.
- UID 1000+ is the range available for assignment to regular users.

Tasks

1. Open an SSH session to localhost as pllab
2. Switch to root using **sudo,** converting to the **root** user's shell environment.
3. Create the **operator2** and confirm that it exists in the system
4. Set the password for **operator2** to **redhat**
5. Create the additional users **operator3** and **operator4**. Set their password to **red hat.**
6. Update the **operator2** and **operator3** user accounts to include the **Operator Two** and **Operator Three** comments, respectively. Verify that the comments are successfully added.
7. Delete the **operator4** user along with any personal data of the user. Confirm that the user is successfully deleted.
8. Exit

MANAGING LOCAL GROUP ACCOUNTS

Managing Local Groups

A group must exist before a user can be added to that group. Several command-line tools are used to manage local group accounts.

Creating Groups from the Command Line

- The **groupadd** command creates groups. Without options the **groupadd** command uses the next available GID from the range specified in the **/etc/login.defs** file while creating the groups
- The **-g** option specifies a particular GID of the group to use

  [pllab@localhost ~] **sudo groupadd –g 10000 group01**

  [pllab@localhost ~] **tail /etc/group**

  …output omitted…

  group01:x:10000:
- The **-r** option creates a system group using a GID from the range of valid system GIDs listed in the **/etc/login.defs** file. The SYS_GID_MIN and SYS_GID_MAX configuration items in **/etc/login.defs** define the range of system GIDs.

  [pllab@localhost ~]$ **sudo groupadd -r group02**

  [pllab@localhost ~]$ **tail /etc/group**

  ...output omitted...

  group01:x:10000:

  group02:x:988:

Modifying Existing Groups from the Command Line

- The **groupmod** command changes the properties of an existing group. The **-n** option specifies a new name for the group.

  [pllab@localhost ~]$ **sudo groupmod -n group0022 group02**

[pllab@localhost ~]$ **tail /etc/group**

...output omitted...

group0022:x:988:

- The -g option specifies a new GID.

  [pllab@localhost ~]$ **sudo groupmod -g 20000 group0022**

  [pllab@localhost ~]$ **tail /etc/group**

  ...output omitted...

  group0022:x:20000:

Deleting Groups from the Command Line

- The **groupdel** command removes groups.

  [pllab@localhost ~]$ **sudo groupdel group0022**

Changing Group Membership from the Command Line

- The membership of a group is controlled with user management. Use the **usermod -g** command to change a user's primary group.

  [pllab@localhost ~]$ **id user02**

  uid=1006(user02) gid=1008(user02) groups=1008(user02)

  [pllab@localhost ~]$ **sudo usermod -g group01 user02**

  [pllab@localhost ~]$ **id user02**

  uid=1006(user02) gid=10000(group01) groups=10000(group01)

- Use the usermod -aG command to add a user to a supplementary group.

  [pllab@localhost ~]$ **id user03**

  uid=1007(user03) gid=1009(user03) groups=1009(user03)

  [pllab@localhost ~]$ **sudo usermod -aG group01 user03**

  [pllab@localhost ~]$ **id user03**

  uid=1007(user03) gid=1009(user03) groups=1009(user03),10000(group01)

Tasks

1. Open an SSH session to localhost as pllab
2. Switch to root using **sudo,** inheriting the full environment of the **root** user.
3. Create the **operators** supplementary group with GID of 3000.
4. Create **admin** as an additional supplementary group.
5. Verify that both the **operators** and **supplementary** groups exist.
6. Ensure that the users **operator1, operator2** and **operator3** belong to the group **operators.**
    a. Add operator1, operator2 and operator3 to operators group.
    b. Confirm that the users are successfully added to the group.
7. Ensure that the users **sysadmin1, sysadmin2** and **sysadmin3** belong to the group **admin.** Enable administrative rights for all the group members of **admin**. Verify that any member of **admin** can run administration commands.
    a. Add sysadmin1, sysadmin2 and sysadmin3 belong to admin.
    b. Confirm that the users are successfully added to the group.
    c. Examine **/etc/group** to verify the supplementary group memberships.
    d. Create the **/etc/sudoers.d/admin** file such that the members of admin have full administrative privileges.
    e. Switch to **sysadmin1** and verify that you can run a **sudo** command as **sysadmin1.**
8. Exit

MANAGING USER PASSWORDS

1. Shadow Passwords and Password Policy

At one time, encrypted passwords were stored in the world-readable /etc/passwd file. This was thought to be reasonably secure until dictionary attacks on encrypted passwords became common. At that point, the encrypted passwords were moved to a separate /etc/shadow file which is readable only by root. This new file also allowed password aging and expiration features to be implemented.

Like **/etc/passwd,** each user has a line in the /etc/shadow file. A sample line from **/etc/shadow** with its nine colon-separated fields is shown below

```
❶user03:❷$6$CSsX...output omitted...:❸17933:❹0:❺99999:❻7:❼2:❽18113:❾
```

1. Username of the account this password belongs to.
2. The encrypted password of the user.
3. The day on which the password was last changed. This is set in days since 1970-01-01, and is calculated in the UTC time zone.
4. The minimum number of days that have to elapse since the last password change before the user can change it again.
5. The maximum number of days that can pass without a password change before the password expires. An empty field means it does not expire based on time since the last change.
6. Warning period. The user will be warned about an expiring password when they login for this number of days before the deadline.
7. Inactivity period. Once the password has expired, it will still be accepted for login for this many days. After this period has elapsed, the account will be locked.
8. The day on which the password expires. This is set in days since 1970-01-01, and is calculated in the UTC time zone. An empty field means it does not expire on a particular date.
9. The last field is usually empty and is reserved for future use.

Format of an Encrypted Password

The encrypted password field stores three pieces of information: the hashing algorithm used the salt, and the encrypted hash. Each piece of information is delimited by the $ sign.

```
$1 6$2 CSsXcYG1L/4ZfHr/$3 2W6evvJahUfzfHpc9X.45Jc6H30E...output omitted...
```

1. The hashing algorithm used for this password. The number 6 indicates it is a SHA-512 hash, which is the default in Red Hat Enterprise Linux 8. A 1 would indicate MD5, a 5 SHA-256.
2. The salt used to encrypt the password. This is originally chosen at random.
3. The encrypted hash of the user's password. The salt and the unencrypted password are combined and encrypted to generate the encrypted hash of the password.

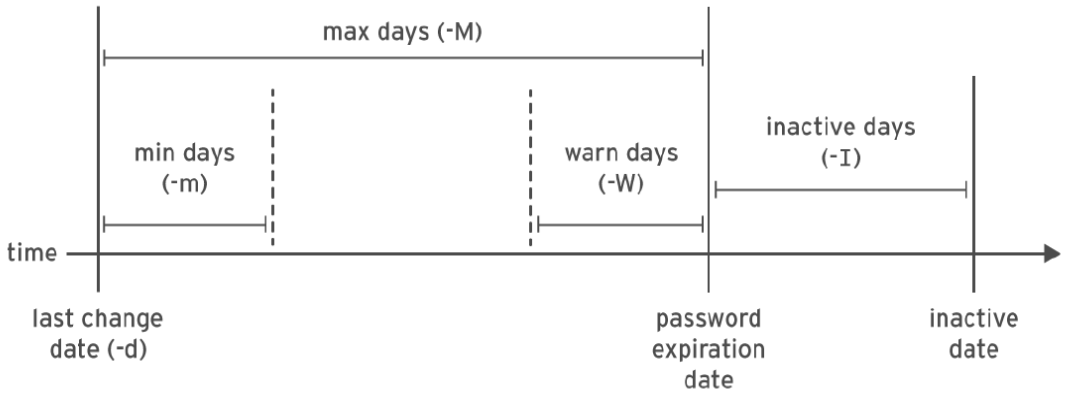
The use of a salt prevents two users with the same password from having identical entries in the **/etc/shadow file**. For example, even if user01 and user02 both use redhat as their passwords, their encrypted passwords in **/etc/shadow** will be different if their salts are different.

Password Verification

When a user tries to log in, the system looks up the entry for the user in **/etc/shadow**, combines the salt for the user with the unencrypted password that was typed in, and encrypts them using the hashing algorithm specified. If the result matches the encrypted hash, the user typed in the right password. If the result does not match the encrypted hash, the user typed in the wrong password and the login attempt fails. This method allows the system to determine if the user typed in the correct password without storing that password in a form usable for logging in.

2. CONFIGURING PASSWORD AGING

The following diagram relates the relevant password aging parameters, which can be adjusted using the chage command to implement a password aging policy.



[pllab@locahost ~]$ **sudo chage -m 0 -M 90 -W 7 -I 14 user03**

- The preceding **chage** command uses the **-m, -M, -W,** and **-I** options to set the minimum age, maximum age, warning period, and inactivity period of the user's password, respectively.
- The **chage -d 0 user03** command forces the user03 user to update its password on the next login.
- The **chage -l user03** command displays the password aging details of user03.
- The **chage -E 2019-08-05 user03** command causes the user03 user's account to expire on 2019-08-05 (in YYYY-MM-DD format).

Edit the password aging configuration items in the /etc/login.defs file to set the default password aging policies. The PASS_MAX_DAYS sets the default maximum age of the password. The PASS_MIN_DAYS sets the default minimum age of the password. The PASS_WARN_AGE sets the default warning period of the password. Any change in the default password aging policies will be effective for new users only. The existing users will continue to use the old password aging settings rather than the new ones.

3. RESTRICTING ACCESS

You can use the **chage** command to set account expiration dates. When that date is reached, the user cannot log in to the system interactively. The usermod command can lock an account with the **-L** option.

[pllab@localhost ~]$ **sudo usermod -L user03**

[pllab@localhost ~]$ **su - user03**

Password: **redhat**

su: Authentication failure

If a user leaves the company, the administrator may lock and expire an account with a single usermod command. The date must be given as the number of days since 1970-01-01, or in the YYYY-MM-DD format.

[pllab@localhost ~]$ **sudo usermod -L -e 2019-10-05 user03**

The preceding **usermod** command uses the -e option to set the account expiry date for the given user account. The **-L** option locks the user's password.

Locking the account prevents the user from authenticating with a password to the system. It is the recommended method of preventing access to an account by an employee who has left the company. If the employee returns, the account can later be unlocked with **usermod -U**. If the account was also expired, be sure to also change the expiration date.

The nologin Shell

The **nologin** shell acts as a replacement shell for the user accounts not intended to interactively log into the system. It is wise from the security standpoint to disable the user account from logging into the system when the user acount serves a responsibility that does not require the user to log into the system. For example, a mail server may require an account to store mail and a password for the user to authenticate with a mail client used to retrieve mail. That user does not need to log directly into the system. A common solution to this situation is to set the user's login shell to /sbin/nologin. If the user attempts to log in to the system directly, the nologin shell closes the connection.

[pllab@locahost ~]$ **usermod -s /sbin/nologin user03**

[pllab@locahost ~]$ **su - user03**

Last login: Wed Feb 6 17:03:06 IST 2019 on pts/0

This account is currently not available.

Tasks

1. Open an SSH session to localhost as **pllab**
2. Explore locking and unlocking user accounts as **pllab**
    a. As **pllab**, lock the **operator1** account using administrative rights
    b. Attempt to login as **operator1.** This should fail.
    c. Unlock the **operator1** account
    d. Attempt to login as **operator1.** This should succeed.
    e. Exit out of the **operator1** user's shell to return to the **pllab** user's shell.
3. Change the password policy for **operator1** to require a new password every 90 days. Confirm that the password age is successfully set.
4. Force the password change on the first login for the **operator1** account
5. Log in as operator1 and change the password to **forsooth123.** After setting the password return to the **pllab,** user's shell.
6. Set the **operator1** account to expire 180days from the current day.
   Hint: The **date -d "+180 days"** gives you the date and time 180 days from the current date and time.
7. Set the passwords to expire 180 days from the current date for all users. Use administrative rights to edit the configuration file.