Chapter 3:
Users and permissions in advanced

Learning objectives

Upon completing this chapter, the learner should be able to:

- Create, delete, and modify local user accounts
- Change passwords and adjust password aging for local user accounts
- Create, delete, and modify local groups and group memberships
- Configure a system to use an existing authentication service for user and group information
- List, set, and change standard ugo/rwx permissions
- Understand and use Linux special permissions
- Understand and use the umask utility
- Create and configure set-GID directories for collaboration
- Diagnose and correct file permission problems

Key terms

User account

System account

root

Primary group

Secondary group

passwd shadow group

File ownership

Permissions

Privileged user

unprivileged user

SUID SGID Stiky bit

Umask

ACL access control list

Table of content

1. Working with users and groups

On Linux, there are two types of users: privileged users, and unprivileged users. The default privileged user is root, known as superuser too, which has full access to everything on a Linux server and is meant to perform system administration tasks and should be used for that only. For all other tasks, an unprivileged user account should be used.

1.1. Working as root

All tasks that involve direct access to devices need root permissions, for example installing software, managing users, and creating partitions on disk devices. For such tasks, root privileges are required and therefore, you should instead use alternative methods like su or sudo tools.

1.1.1. The su tool

The su command allows users to open a terminal window, and from that terminal start a sub shell in which the user has another identity even if it's the root user, this brings the benefit that only in the root shell root privileges are used.

If just the command su is typed, the username root is implied. However, su can be used to run tasks as another user as well. By doing though, you are prompted for the target user password.

```
[user1@server1 ~] $ su
Password:
[root@server1 user1] #
```

When using su, a sub shell is started. This is an environment where you are able to work as the target user account, but environment settings for that user account have not been set. If you need complete access to the entire environment of the target user account, you can use su - to start a login shell. If you start a login shell, all scripts that make up the user environment are processed, which makes you work in an environment that is exactly the same as when logging in as that user.

1.1.2. The sudo tool

Instead of using the root user account, unprivileged users can be configured for using administrator permissions on specific tasks by using sudo. When sudo is configured, ordinary users have sudo privileges and to use these privileges, they will start the command using sudo. So, instead of using commands like useradd as the root user, you use an ordinary user account and type sudo useradd. This is definitely more secure because you will only be able to act as if you have administrator permissions while running this specific command.

When creating Linux users during the installation process, you can select to grant administrator permissions to that specific user. If you select to do so, the user will be able to use all administrator commands using sudo. It is also possible to set up sudo privileges after installation. To do that in a very easy way, you have to accomplish a simple two-step procedure:

- Make the administrative user account member of the group wheel by using usermod -aG wheel user
- Type visudo and make sure the line %wheel ALL=(ALL) ALL is included

Exercise: Switching user accounts

1. Log in to your system as a non-privileged user and open a terminal.
2. Type whoami to see which user account you are currently using. Type id as well, and notice that you get more detail about your current credentials when using id.
3. Type su. When prompted for a password, enter the root password. Type id again. You see that you are currently root.
4. Type visudo and make sure that the line %wheel ALL=(ALL) ALL is included.
5. Type useradd -G wheel lisa to create a user lisa who is a member of the group wheel.
6. Type id lisa to verify that she has been added to the group wheel.
7. Set the password for lisa by typing passwd lisa. Enter the password password twice.
8. Log out and log in as lisa.
9. Type sudo useradd lori. Enter the password when asked. You notice that user lori will be created.

1.2. Managing user properties

On a typical Linux environment, two kinds of user accounts exist:

- User accounts, for the people who need to work on a server and who need limited access to the resources on that server. These user accounts typically have a password that is used for authenticating the user to the system.
- System accounts, which are used by the services the server is offering.

Both user accounts share common properties, which are kept in the files /etc/passwd and /etc/shadow.

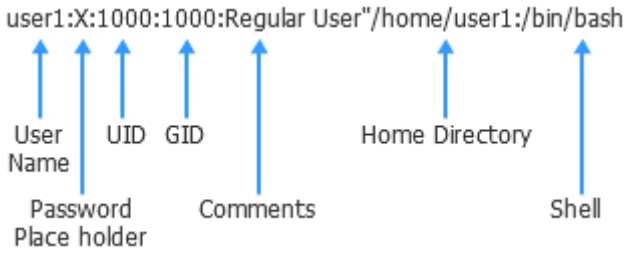
1.2.1. /etc/passwd

Listing the content of the file will show something similar to this:

```
[user1@server1 ~]$ cat /etc/passwd
root:x:0:0::/root:/bin/bash


postfix:x:89:89::/var/spool/postfix:/sbin/nologin
sshd:x:74:74:Privilege-separated
SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
user1:x:1000:1000:User 1:/home/user:/bin/bash
```

As you can see, to define a user account different fields are used, these fields are separated from each other by a colon.



- **Username**: This is a unique name for the user. User names are important to match a user to his password, which is stored separately in /etc/shadow. On Linux, there can be no spaces in the user name.
- **Password**: In the old days, the second field of /etc/passwd was used to store the hashes password of the user. Because the /etc/passwd file is readable by all users, this poses a security threat, and for that reason on current Linux systems the hashes passwords are stored in /etc/shadow (discussed in the next section).
- **UID**: Each user has a unique user ID (UID). This is a numeric ID. It is the UID that really determines what a user can do. When permissions are set for a user, the UID is stored in the file metadata (and not the user name). UID 0 is reserved for root, the unrestricted user account. The lower UIDs (typically up to 999) are used for system accounts, and the higher UIDs (from 1000 on by default), are reserved for people that need to connect directly to the server. The range of UIDs that are used to create regular user accounts is set in /etc/login.defs.

- **GID**: On Linux, each user is a member of at least one group. This group is referred to as the primary group, and this group plays a central role in permissions management, as discussed later in this chapter.
- **Comment field**: The Comment field, as you can guess, is used to add comments for user accounts. This field is optional, but it can be used to describe what a user account is created for.
- **Directory**: This is the initial directory where the user is placed after logging in, also referred to as the home directory. If the user account is used by a person, this is where the person would store his personal files and programs. For a system user account, this is the environment where the service can store files it needs while operating.

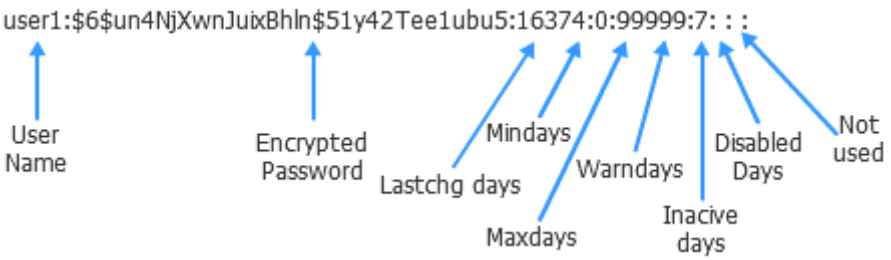
1.2.2. /etc/shadow

The settings in this file are used to define the validity of the password. Typical for /etc/shadow is that no one except the superuser root and processes running with root permissions have permissions to access it, such as the authentication mechanisms on the server.

```
[root@server1 ~]# cat /etc/shadow
root:$6$3VZbGx1djo6FfyZo$/Trg7Q.3foIsIFYxBm6UnHuxxBrxQxHDnDuZxgS.We/
MAuHn8HboBZzpaMD8gfm.fmlB/ML9LnuaT7CbwVXx31:16420:0:99999:7:::
ntp:!!:16420::::::


sshd:!!:16420::::::
tcpdump:!!:16420::::::
user1:$6$3VZbGx1djo6FfyZo$/Trg7Q.3foIsIFYxBm6UnHuxxBrxQxHDnDuZxgS.We/
MAuHn8HboBZzpaMD8gfm.fmlB/ML9LnuaT7CbwVXx31:16420:0:99999:7:::
```

The following fields are defined in `/etc/shadow`:



- **Login name**: Notice that `/etc/shadow` does not contain any UIDs, but usernames only. This opens a possibility for multiple users using the same UID but different passwords (which, by the way, is not really recommended).
- **Encrypted password**: This field contains all that is needed to store the password in a secure way. Days since Jan 1, 1970, that the password was last changed: Many things on Linux refer to this date, which on Linux is considered the beginning of days. It is also referred to as epoch.
- **Days before password may be changed**: This allows system administrators to use a more strict password policy, where it is not possible to change back to the original password immediately that a password has been changed. Typically, this field is set to the value 0.

- **Days after which password must be changed**: This field contains the maximal validity period of passwords. Notice that by default it is set to 99,999 (about 273 years).
- **Days before password is to expire that user is warned**: This field is used to warn a user when a forced password change is upcoming. Notice that the default is set to 7 (even if the password validity is set to 99,999 days!).
- **Days after password expires that account is disabled**: Use this field to enforce a password change. After password expiry, users can log in no longer.
- **Days since Jan 1, 1970, that account is disabled**: An administrator can set this field to disable an account. This is typically a better approach than removing an account, as all associated properties and files of the account will be kept, but it can be used no longer to authenticate on your server.
- **A reserved field, which was once added "for future use"**: That was a long time ago, it will probably never be used.

1.2.3. Tools

Command	Description
useradd	Adds a user account by using many of its parameters. **-r**: creates a system user. **-c**: sets the user comment field. **-e**: for the date on which the user account will be disabled. **-a**: adds the user to the supplementary group(s). **-G**: a list of supplementary groups the user is also a member of.
usermod	Modifies a user account by using many of its parameters, mostly, the useradd parameters are used here too. **-l**: changes the login name of the user. **-p**: sets the encrypted password. **-m**: moves the content of the user's home directory to the new location.
userdel	Removes user, -r option to remove the user's environment too.
vipw	This command opens an editor interface on your /etc/passwd file, and it sets the appropriate locks on the configuration files to prevent corruption.

passwd	This command changes passwords for user accounts, also changes the account or associated password validity period. **-d**: deletes a user's password, it will set the named account passwordless. **-e**: expires a user's account password. **-l**: locks the password of a user, but he/she may still be able to login using another authentication token (e.g. an SSH key). **-n**: sets the minimum number of days between password changes. **-w**: sets the number of days of warning before a password change is required. **-x**: sets the maximum number of days a password remains valid.
change	change user password expiry information. **-l**: shows account aging information. **-E**: sets the date on which the user's account will no longer be accessible. **-m**: sets the minimum number of days between password changes. **-M**: sets the maximum number of days during which a password is valid. **-W**: changes user password expiry information.
vipw -s	This command opens an editor interface on your /etc/shadow file, and it sets the appropriate locks on the configuration files to prevent corruption.

1.3. Understanding user environment

When a user logs in, an environment is created. The environment consists of some variables that determine how the user environment is used like the $PATH variable. To construct the user environment, a few files play a role:

- /etc/profile: Used for default settings for all users when starting a login shell
- /etc/bashrc: Used to define defaults for all users when starting a subshell
- ~/.profile: Specific settings for one user applied when starting a login shell
- ~/.bashrc: Specific settings for one user applied when starting a subshell

When logging in, the files are read in this order, and variables and other settings that are defined in these files are applied. If a variable or setting occurs in more than one file, the last one wins.

Home Directories

All normal users will have a home directory. For system accounts, the home directory often contains the working environment for the service account and are created automatically from the RPM post installation scripts when installing the related software packages.

For people, the home directory is the directory where personal files can be stored. When creating user accounts, you tell your server to add a home directory as well (for instance, by using useradd -m), the content of the “skeleton” directory is copied to the user's home directory. The skeleton directory is /etc/skel, and it contains files that are copied to the user home directory at the moment this directory is created.

These files will also get the appropriate permissions to ensure that the new user can use and access them. By default, the skeleton directory contains mostly configuration files that determine how the user environment is set up. If in your environment specific files need to be present in the home directories of all users, you take care of that by adding the files to the skeleton directory.

1.4. User management defaults

When working with tools as useradd, some default values are assumed. These default values are set in two configuration files: /etc/login.defs and /etc/default/useradd.

```
[root@server1 skel] # cat /etc/default/useradd
# useradd defaults file
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
CREATE_MAIL_SPOOL=yes
```

As shown above, the /etc/default/useradd file contains some default values that are applied when using useradd.

In the file /etc/login.defs, different login-related variables are set. This file is used by different commands, and it relates to setting up the appropriate environment for new users. Here is a list of some of the most significant properties that can be set from /etc/login.defs:

- **MOTD_FILE**: Defines the file that is used as "message of the day" file. In this file, you can include messages to be displayed after the user has successfully logged in to the server.
- **ENV_PATH**: Defines the $PATH variable, a list of directories that should be searched for executable files after logging in.
- **PASS_MAX_DAYS, PASS_MIN_DAYS, and PASS_WARN_AGE**: Define the default password expiration properties when creating new users.
- **UID_MIN**: The first UID to use when creating new users.
- **CREATE_HOME**: Indicates whether or not to create a home directory for new users.
- **USERGROUPS_ENAB**: Set to yes to create a private group for all new users. That means that a new user has a group with the same name as the user as its default group. If set to no, all users are made a member of the group users.

Exercise: Creating user accounts

1. Type vim /etc/login. defs to open the configuration file /etc/login. defs and change a few parameters before you start creating logging. Look for the parameter CREATE_HOME and make sure it is set to “yes.” Also set the parameter USERGROUPS_ENAB to “no,” which makes that a new user is added to a group with the same name as the user and nothing else.
2. Use cd /etc/skel to go to the /etc/skel directory. Type mkdir Pictures and mkdir Documents to add two default directories to all user home directories. Also change the contents of the file. bashrc to include the line export EDITOR = /usr/bin/vim, which sets the default editor for tools that need to modify text files.
3. Type useradd linda to create an account for user linda. Then, type id linda to verify that linda is a member of a group with the name linda and nothing else. Also verify that the directories Pictures and Documents have been created in linda’s home directory.
4. Use passwd linda to set a password for the user you have just created. Use the password password.
5. Type passwd -n 30 -w 3 -x 90 linda to change the password properties. This has the password expire after 90 days (-x 90). Three days before expiry, the user will get a warning (-w 3), and the password has to be used for at least 30 days before (-n 30) it can be changed.
6. Create a few more users: lisa, lori, and bob, using for i in lisa lori bob, do useradd $i, done.
7. Use grep lori /etc/passwd /etc/shadow /etc/group. This shows the user lori created in all three critical files and confirms they have been set up correctly.

1.5. Managing group properties

Every Linux user has to be a member of at least one group, and they can be a member of two different kinds of groups. First, there is the primary group, every user must be a member of a primary group and there is only one primary group. When creating files, the primary group becomes group owner of these files.

Users can also access all files their primary group has access to. The users primary group membership is defined in /etc/passwd, the group itself is stored in the /etc/group configuration file.

Besides the mandatory primary group, users can be a member of one or more secondary groups as well. Secondary groups are important to get access to files, where if the group a user is a member of has access to specific files, the user will get access to these files also. Working with secondary groups is important, in particular in environments where Linux is used as a file server to allow people working for different departments to share files with one another.

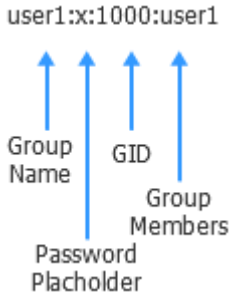
1.5.1. /etc/group

Listing the content of the file will show something similar to this:

```
[user1@server1 ~] $ cat /etc/group
root: x: 36: root


kvm: x: 36: qemu
qemu:x:107:
libstoragemgmt:x:994:
rpc:x:32:
user1:x:1000:user1
```

The following fields are used in /etc/group:



- **Group name:** As is suggested by the name of the field, this contains the name of the group
- **Group password:** A feature that is hardly used anymore. A group password can be used by users that want to join the group on a temporary basis
- **Group ID:** A unique numeric group identification number
- **Members:** Here you find the names of users that are a member of this group as a secondary group. Note that it does not show users that are a member of this group as their primary group

1.5.2. Tools

Here is a list of the main tools used to manage and control the group accounts in Linux systems.

Command	Description
groupadd	Creates a new group. **-r**: creates a system group. **-p**: sets the encrypted password. **-g**: sets the numerical value of the group's ID.
groupmod	Modifies a group definition on the system. **-n**: sets new groups name.
groupdel	Deletes a group.
vigr	Opens an editor interface directly on the /etc/group configuration file, and it sets the appropriate locks on the configuration files to prevent corruption.
groupmems	Administers members of a user's primary group. **-a**: adds an user to the group membership list. **-d**: deletes a user from the group membership list. **-g**: specifies which group membership list to modify. **-l**: lists the group membership list. **-p**: purges all users from the group membership list.

Exercise: Working with groups

In this exercise, you create two groups and add some users as members to these groups.

1. Type groupadd sales followed by groupadd account to add groups with the names sales and account.

2. Use usermod to add users linda and lisa to the group sales, and lori and bob to the group account:

   usermod -aG sales linda

   usermod -aG sales lisa

   usermod -aG account lori

   usermod -aG account bob

3. Type id linda to verify that user linda has correctly been added to the group sales. In the results of this command, you see that linda is assigned to the group with gid = 100(users). This is her primary group. With the groups parameter, all groups she is a member of as secondary group are mentioned:

   linux: ~ # id linda

   uid=1000 (linda) gid=100 (users) groups=1000 (sales), 100 (users)

2. Managing advanced permissions

2.1. File ownership

File and directory ownership are vital for working with permissions, where on Linux, every file and every directory has two owners: a user and a group owner. On creation, the user who creates the file becomes the user owner, and the primary group of that user becomes the group owner. The shell checks ownership in the following order:

1. The shell checks whether you are the user owner of the file you want to access, which is also referred to as the user of the file. If you are the user, you get the permissions that are set for the user, and the shell looks no further.
2. If you are not the user owner, the shell will check whether you are a member of the group owner, which is also referred to as the group of the file. If you are a member of the group, you get access to the file with the permissions of the group, and the shell looks no further.
3. If you are neither the user, nor the group owner, you get the permissions of others.

Tools like `ls -l` is used to show the user and the group owner for files in a given directory. Also `find` tool is useful to get a list of all files on the system that have a given user or group as owner, this is done by using the `-user` and `-group` arguments like this: `find /etc-user root-group root`.

2.1.1. Changing ownership

To apply appropriate permissions, the first thing to consider is ownership, there are the chown and chgrp commands, the syntax of this command is not hard to understand: chown who what.

Where the table below shows basic usage of those commands.

Command	Description
chown user1 file1	Sets user user1 as the owner of file1
chown user1. group1 file1 chown user1: group1 file1	Sets user user1 as user owner and group group1 as group owner of file1
chown. group1 file1 chown: group1 file1	Sets group group1 as group owner of file1 without changing the user owner
chgrp group1 file1	Sets group group1 as group owner of file1 without changing the user owner
chown –R, chgrp –R	to change ownership recursively

2.1.2. Default ownership

Default ownership is applied when files or directories are created, the user who creates the file automatically becomes user owner, and the primary group of that user automatically becomes group owner. If the user is a member of more groups, however, he can change the effective primary group by using the newgrp command followed by the name of the group to be set as the new effective primary group. This group will be continued to be used as effective primary group until the user uses the exit command or logs out.

```
[user1@server1 ~] $ groups
user1 account sales
[user1@server1 ~] $ newgrp sales
[user1@server1 ~] $ groups
sales lisa account
[user1@server1 ~] $ touch file1
[user1@server1 ~] $ ls -l
total 0
-rw-r--r--. 1 lisa sales 0 Feb 6 10:06 file1
```

To be able to use the newgrp command, a user has to be a member of that group. Alternatively, a group password can be set for the group using the gpasswd command.

2.2. Basic permissions review

The table below summarizes the use of the basic permissions.

Permission	On files	On directories
Read	Open a file	List contents of directory
Write	Change the contents of a file	Create and delete files and modify permissions on files
Execute	Run a program file	Change to the directory

Applying Read, Write, and Execute Permissions

To apply permissions, you use the chmod command to set permissions for user, group, and others. You can use this command in two modes: the relative mode and the absolute mode.

In absolute mode, three digits are used to set the basic permissions, the table below provides an overview of the permissions:

Permission	Numeric Representation
Read	4 = 2^2
Write	2 = 2^1
Execute	1 = 2^0

When you use chmod in this way, all current permissions are replaced by the permissions you set.

If you want to set read, write, and execute for the user, read and execute for the group, and read and execute for others on the file /somefile, for example, you use the following chmod command:

```
chmod 755 /somefile
```

In relative mode, three indicators to specify what you want to do:

- First, you specify for whom you want to change permissions. To do this, you can choose between user (u), group (g), and others (o)
- Then, you use an operator to add or remove permissions from the current mode, or set them in an absolute way
- At the end, you use r, w, and x to specify what permissions you want to set

When changing permissions in relative mode, you may omit the “to whom” part to add or remove a permission for all entities. For instance, the following adds the execute permission for all users:

```
chmod+x somefile
```

When working in relative mode, you may use more complex commands as well. For instance, the following adds the write permission to the group and remove read for others:

```
chmod g+w, o-r somefile
```

When using `chmod -R o+rx /data`, you set the execute permission on all directories as well as files in the /data directory. To set the execute permission to directories only, and not to files, use `chmod -R o+rx /data`. The uppercase X ensures that files will not get the execute permission unless the file has already set the execute permission for some of the entities. That makes X the more intelligent way of dealing with execute permissions; it will avoid setting that permission on files where it is not needed.

Exercise: Managing basic permissions

In this exercise, you create a directory structure for the groups and also assign the correct permissions to these directories.

1. From a root shell, type mkdir -p /data/sales /data/account.
2. Before setting the permissions, change the owners of these directories using chown linda. sales /data/sales and chown linda. account /data/account.
3. Set the permissions to enable the user and group owners to write files to these directories, and deny all access for all others: chmod 770 /data/sales, and next chmod 770 /data/account.
4. Use su - lisa to become lisa and change into the directory /data/account. Use touch emptyfile to create a file in this directory. Does this work?
5. Still as lisa, use cd /data/sales and use touch emptyfile to create a file in this directory. Does this work?

2.3. Working with special permissions

Apart from the basic permissions, Linux has a set of advanced permissions as well which sometimes provide a useful addition.

2.3.1. Set User ID

Known as Set User ID (SUID) permission, it's applied to executable files. By default, a user who runs an executable file runs this file with his own permissions. In some cases, however, the user needs special permissions, just for the execution of a certain task, for example, the situation where a user needs to change his password. To do this, the user needs to write his new password to the /etc/shadow file. This file, is not writeable for users who do not have root permissions:

```
[root@hnl ~]# ls -l /etc/shadow
----------. 1 root root 1184 Apr 30 16:54 /etc/shadow
```

The SUID permission offers a solution for this problem. On the /usr/bin/passwd utility, this permission is applied by default. That makes the user temporarily has root permissions, which allows him to write to the /etc/shadow file. You can see the SUID permission with ls -l as an s at the position where normally you would expect to see the x for the user permissions:

```
[root@hnl ~] # ls -l /usr/bin/passwd
-rwsr-xr-x. 1 root root 32680 Jan 28 2010 /usr/bin/passwd
```

The SUID permission may look useful (and it is in some cases), but at the same time, it is potentially dangerous. If applied incorrectly, you may give away root permissions by accident. Most administrators will never have to use it, you'll only see it on some files where the operating system needs to set it as a default.

2.3.2. Set Group ID

Stands for Set Group ID (SGID), this permission has two effects related on what it's applied. If applied on an executable file, it gives the user who executes the file permissions of the group owner of that file. For this purpose, however, SGID is hardly used, as is the case for the SUID permission, SGID is applied to some system files as a default setting.

When applied to a directory it sets the default group ownership on files and subdirectories created in that directory. By default, when a user creates a file, his effective primary group is set as the group owner for that file. That means files that a user creates will be group shared with nobody else.

If you create a shared group directory (say, /groups/account) and make sure that the SGID permission is applied to that directory, and that the group accounting is set as the group owner for that directory, all files created in this directory and all its subdirectories also get the group accounting as the default group owner. For that reason, the SGID permission is a very useful permission to set on shared group directories.

The SGID permission shows in the output of ls -l as an s at the position where you normally find the group execute permission:

```
[root@hnl data]# ls -ld account
drwxr-sr-x. 2 root account 4096 Apr 30 21:28 account
```

2.3.3. Sticky bit

This permission is useful to protect files against accidental deletion in an environment where multiple users have write permissions in the same directory. If sticky bit is applied, a user may delete a file only if he is the user owner of the file or the directory that contains the file. It is for that reason applied as a default permission to the /tmp directory, and it can be useful on shared group directories as well.

Without sticky bit, if a user can create files in a directory, he can also delete files from that directory. In a shared group environment, this may be annoying. When you apply the sticky bit, a user can delete files only if either of the following is true:

- The user is the owner of the file
- The user is the owner of the directory where the file exists

When using ls -l, you can see sticky bit at the position where you normally see the execute permission for others:

```
[root@hnl data]# ls -ld account/
drwxr-sr-t. 2 root account 4096 Apr 30 21:28 account/
```

2.3.4. Applying special permissions

To apply SUID, SGID, and sticky bit, you can use chmod as well. SUID has numeric value 4, SGID has numeric value 2, and sticky bit has numerical value 1.

If you want to apply these permissions, you need to add a four-digit argument to chmod, of which the first digit refers to the special permissions. For example, chmod 2755 /somedir adds the SGID permission to a directory, and set rwx for user and rx for group and others:

It's recommended to work in relative mode if you need to apply any special permissions:

- For SUID, use chmod u+s,
- For SGID, use chmod g+s,
- For sticky bit, use chmod +t, followed by the name of the file or the directory that you want to set the permissions on

The table below summarizes the special permissions usage and effect:

Permission	On files	On directories
SUID (4) u+s	User executes file with permissions of file owner.	No meaning.
SGID (2) g+s	User executes file with permissions of group owner.	Files created in directory get the same group owner.
Stiky bit (1) o+t_0	No meaning.	Prevents users from deleting files from other users.

2.4. Working with umask

When creating a new file, some default permissions are set and determined by the umask (User Mask) setting, this shell setting is applied to all users when logging in to the system.

In the umask setting, a numeric value is used that is subtracted from the maximum (initial) permissions that can be set automatically to a file, the maximum setting for files is 666 (rw–rw–rw–), and for directories is 777 (rwxrwxrwx).

The first digit of umask value refers to user permissions, the second digit refers to the group permissions, and the last refers to default permissions set for others. Default umask setting is:

- 0022 for the root and other system users, which gives 644 for all new files and 755 for all new directories that are created on your server
- 0002 for all regular users, which gives 664 for all new files and 775 for all new directories
- User commands like umask and umask -S to see the current umask value in the system

A complete overview of all umask numeric values and their result is shown in Table below.

Value	On files	On directories
0	Read and write	Everything
1	Read and write	Read and write
2	Read	Read and execute
3	Read	Read
4	Write	Write and execute
5	Write	Write
6	Nothing	Execute
7	Nothing	Nothing

Now, if you wish to have different default permissions set for new files and directories, you need to modify the umask. You first need to determine the desired default values. For instance, if you want all your new files and directories to get 640 and 750 permissions, respectively, you can set the value to 027 as follows: `umask 027`.

The new value becomes effective right away, and it will only be applied to files and directories created thereafter. The existing files and directories will remain intact.

The umask setting is considered when starting the shell environment files as directed by /etc/profile, so there are two ways to change the umask setting: for all users and for individual users.

- **For all users**: The right approach is to create a shell script with the name umask.sh in the /etc/ profile. d directory and specify the umask you want to use in that shell script. If the umask is changed in this file, it applies to all users after logging in to your server.
- **For individual users**: change the umask settings in a file named. profile, which is created in the home directory of an individual user. Settings applied in this file are applied for the individual user only, therefore, this is a nice method if you need more granularity.

Exercise: Working with special permissions

In this exercise, you use special permissions to make it easier for members of a group to share files in a shared group directory. You assign the set group ID bit, as well as sticky bit, and see that after setting these, features are added that make it easier for group members to work together.

1. Open a terminal in which you are user linda.
2. Use cd /data/sales to go to the sales directory. Use touch linda1 and touch linda2 to create two files of which linda is the owner.
3. Use su - lisa to switch the current user identity to user lisa, who also is a member of the sales group.
4. Use cd /data/sales and from that directory, use ls -l. You'll see the two files that were created by user linda that are group-owned by the group linda. Use rm -f linda*. This will remove both files.
5. Use the commands touch lisa1 and touch lisa2 to create two files that are owned by user lisa.
6. Use su - to escalate your current permissions to root level.
7. Use chmod g+s, o+t /data/sales to set the group ID bit as well as sticky bit on the shared group directory.
8. Use su - linda. First, use touch linda3 and touch linda4. You should now see that the two files you have created are owned by the group sales who is group owner of the directory /data/sales.
9. Use rm -rf lisa*. Sticky bit prevents you from removing these files as user linda because you are not the owner of the files. Note that if user linda is directory owner of /data/sales, she can remove the files in question anyway!

Quiz

Chapter review questions

1. Which statement about privileged users (root) is true?

    a. A privileged user is a user who has access to a Linux system.

    b. A privileged user with no access permissions can do nothing at all.

    c. Privileged users are not restricted in any way.

    d. On every server, at least one privileged user must be manually created while installing the server.

2. On a default installation of a RHEL 7 server, which group does the user typically need to be a member of to be able to use sudo to run all administration commands?

    a. admin

    b. root

    c. sys

    d. wheel

3. There are different ways that users can run tasks with root permissions. Which of the following is not one of them?

    a. sudo

    b. runas

    c. su

    d. PolicyKit

4. Which of the following is used to store the hash of the user's encrypted password?

    a. /etc/passwd

    b. /etc/shadow

    c. /etc/users

    d. /etc/secure

5. Which configuration file should you change to set the default location for all new user home directories?

    a. /etc/login.defaults

    b. /etc/login.defs

    c. /etc/default/useradd

    d. /etc/default/login.defs

6. Which command enables you to get information about password properties such as password expiry?

    a. chage –l

    b. usermod – show

    c. passwd –l

    d. chage ––show

7. Which of the following files is not processed when a user starts a login shell?

    a. /etc/profile

    b. /etc/. profile

    c. ~/. bashrc

    d. ~/. bash_profile

8. A user who is a member of several groups wants to change default group ownership for all newly created files and set it to the group sales. Which command would do that?

    a. chgrp sales

    b. setgid sales

    c. newgrp sales

    d. setgroup sales

9. Which command enables you to find all files on a system that are owned by user linda?

    a. find / –user linda

    b. find / –uid linda

    c. ls –l | grep linda

    d. ls –R | find linda

10. Which of the following commands does not set group ownership to the group sales for the file my file?

    a. chgrp sales my file

    b. chown. sales my file

    c. chgrp my file sales

    d. chown: sales my file

11. Which of the following would be used to allow read and write permissions to the user and group owners and no permissions at all to anyone else?

a. chown 007 filename

b. chmod 077 filename

c. chmod 660 filename

d. chmod 770 filename

12. Which command enables you to set the SGID permission on a directory?

a. chmod u+s/dir

b. chmod g-s/dir

c. chmod g+s/dir

d. chmod 1770/dir

13. You are trying to use the setfacl command to set ACLs on the directory /data, but you are getting an "operation not supported" message. Which of the following is the most likely explanation?

a. The setfacl command is not installed on your computer.

b. You are making an error typing the command.

c. The user or group to which you want to grant ACLs does not exist.

d. The file system lacks ACL support.

14. Which of the following umask settings meets the following requirements?

- Grants all permissions to the owner of the file.
- Grants read permissions to the group owner of the file.
- Grants no permissions to others.

a. 740

b. 750

c. 027

d. 047

15. What is the difference between running the su command with and without the dash sign?

a. su starts a login shell, but su – starts a subshell.

b. su starts a subshell, but su – starts a login shell.

c. su makes you work in an environment that is exactly the same as when logging in as that user, but su – doesn't do that.

d. None of the above.

16. What is the utility for manually editing shadow password files exclusively?

a. vipw

b. vigr

c. viur

d. viot

17. What does the “x” in the password field in the passwd file imply?

a. The encrypted password is stored in the shadow file.

b. There is no password for this user.

c. The encrypted password is stored in the goup file.

d. The encrypted password is not encrypted.

18. The passwd file contains secondary user group information.

a. True

b. False

19. What is the tool for changing shadow password?

a. The passwd.

b. The vipw utility.

c. The groups utility.

d. The vipasswd utility.

20. What is the name and location of the sudo configuration file?

a. /etc/sudoers

b. /etc/sudo

c. /usr/local/sudors

d. /etc/sudor

Answers to chapter Review Questions:

1. d
2. d
3. b
4. b
5. a
6. a
7. b
8. b
9. a
10. c
11. c
12. c
13. d
14. a
15. b
16. a
17. a
18. b
19. a
20. a