Objective 1 Understand User and Group Configuration Files

Information on users and groups on a Linux system is kept in the following files:

- /etc/passwd
- /etc/shadow
- /etc/group

Whenever possible, you should not modify these files with an editor. Instead, use the Security and Users modules in YaST or the command line tools described in the next objective, "Manage User Accounts and Groups from the Command Line" on 7-12.

Modifying these files with an editor can lead to errors (especially in /etc/shadow), such as a user—including the user root—no longer being able to log in.

To ensure consistency of these files, you need to understand how to

- Check /etc/passwd and /etc/shadow
- Convert Passwords to and from Shadow

/etc/passwd

The file **/etc/passwd** stores information for each user. In the past, UNIX and Linux users were handled in a single file: /etc/passwd. The user name, the UID, the home directory, the standard shell, and the encrypted password were all stored in this file.

The password was encrypted using the function crypt (man 3 crypt). In principle, the plain text password could not be deciphered from the encrypted password.

However, there are programs (such as john) that use dictionaries to encrypt various passwords with crypt, and then compare the results with the entries in the file /etc/passwd.

With the calculation power of modern computers, simple passwords can be “guessed” within minutes.

The main problem with the file /etc/passwd is that it has to be readable by any user. Because only the UID is saved in the inode of a file, /etc/passwd is used to map UIDs to user names.

The logical solution to this problem has been to store the password field in its own file (/etc/shadow), which can only be read by root.

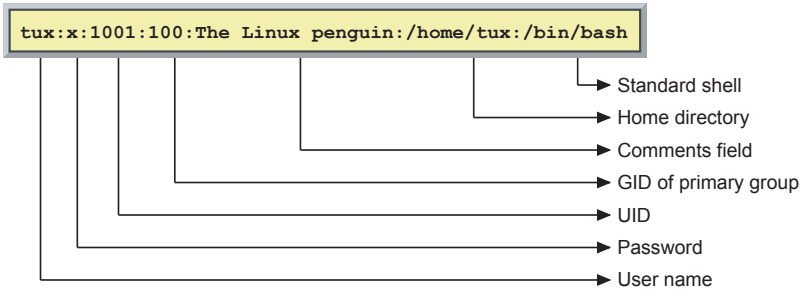
The following is a sample /etc/passwd file:

Figure 7-1

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
daemon:x:2:2:Daemon:/sbin:/bin/bash
lp:x:4:7:Printing daemon:/var/spool/lpd:/bin/bash
mail:x:8:12:Mailer daemon:/var/spool/clientmqueue:/bin/false
news:x:9:13:News system:/etc/news:/bin/bash
uucp:x:10:14:Unix-to-Unix CoPy system:/etc/uucp:/bin/bash
games:x:12:100:Games account:/var/games:/bin/bash
man:x:13:62:Manual pages viewer:/var/cache/man:/bin/bash
at:x:25:25:Batch jobs daemon:/var/spool/atjobs:/bin/bash
postgres:x:26:26:PostgreSQL Server:/var/lib/pgsql:/bin/bash
mdom:x:28:28:Mailing list agent:/usr/lib/majordomo:/bin/bash
wwwrun:x:30:8:WWW daemon apache:/var/lib/wwwrun:/bin/false
squid:x:31:65534:WWW-proxy squid:/var/cache/squid:/bin/false
amanda:x:37:6:Amanda admin:/var/lib/amanda:/bin/bash
irc:x:39:65534:IRC daemon:/usr/lib/ircd:/bin/bash
ftp:x:40:49:FTP account:/srv/ftp:/bin/bash
named:x:44:44:Name server daemon:/var/lib/named:/bin/false
gdm:x:50:15:Gnome Display Manager daemon:/var/lib/gdm:/bin/bash
geeko:x:1000:100:geeko:/home/geeko:/bin/bash
tux:x:1001:100:The Linux Penguin:/home/tux:/bin/bash
```

Each line in the file /etc/password represents one user, and contains the following information:

Figure 7-2



Note the following about the fields in each line:

- **User name.** This is the name a user enters to log in to the system (login name).

  Although Linux can handle longer user names, in this file they should be restricted to a maximum of eight characters for backward compatibility with older programs.

- **Password.** The x in this field means that the password is stored in the file /etc/shadow.

- **UID.** In compliance with the Linux standards, two number ranges are reserved:

  - **0–99** for the system itself
  - **100–499** for special system users (such as services and programs)

  On SLES 9, normal users start from UID 1000.

- **Comments field.** Normally, the full name of the user is stored here. Information such as a room number or telephone number can be entered as well.

- **Home directory.** The personal directory of a user is normally in the directory /home/ and has the same name as the user (login) name.
- **Standard shell.** This is the shell that is started for a user after he has successfully logged in. In Linux this is normally bash.

  The shell must be listed in the file **/etc/shells**. Each user can change her standard shell with the command chsh (see **man chsh**).

For additional information on this file, enter **man 5 passwd**.

/etc/shadow

The **/etc/shadow** file stores encrypted user passwords and password expiration information. Most Linux systems use **shadow passwords**. Shadow passwords are stored in /etc/shadow instead of /etc/passwd.

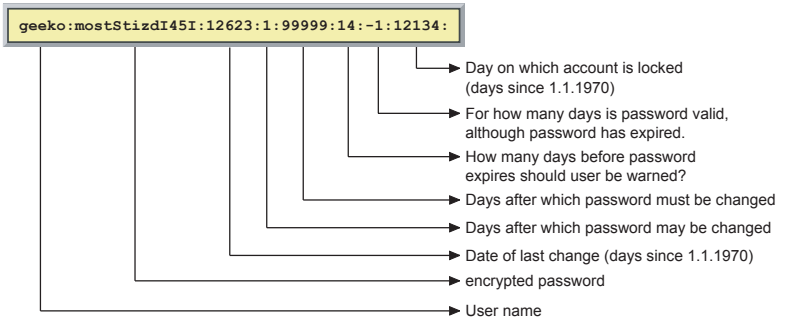
This file can only be changed by the user root and read by the user root and members of the group shadow. The following is a sample /etc/shadow file:

Figure 7-3

```
mailman:!:12608:0:99999:7:::
man:*:8902:0:10000::::
mdom:!:12 08:0:99999:7:::
mysql:!:12608:0:99999:7:::
named:!:12608:0:99999:7:::
news:*:8902:0:10000::::
nobody:*:8902:0:10000::::
ntp:!:12608:0:99999:7:::
pop:!:12608:0:99999:7:::
postfix:!:12608:0:99999:7:::
postgres:!:12608:0:99999:7:::
quagga:!:12608:0:99999:7:::
radiusd:!:12608:0:99999:7:::
root:XOQeyibhsgHj2:12608:0:10000::::
snort:!:12608:0:99999:7:::
squid:!:12608:0:99999:7:::
sshd:!:12608:0:99999:7:::
stunnel:!:12608:0:99999:7:::
uucp:*:8902:0:10000::::
vscan:!:12608:0:99999:7:::
wwwrun:*:8902:0:10000::::
tux:svSIYQsFoEwKg:12608:0:99999:7:-1::
geeko:mostSt1zdI45I:12623:1:99999:14:-1:12134:
```

Each line in the file /etc/shadow belongs to one user and contains the following fields:

Figure 7-4



This figure shows the entry for the user **geeko** with his encrypted password. (Technically, it is more correct to speak of a hashed password.)

The encrypted password is coded with the crypt function and is always 13 characters long. The encrypted word consists of letters, numbers, and the special characters . and /.

If an invalid character occurs in the password field (such as * or !), then the user cannot log in.

Many users, such as wwwrun (Apache Web server) or bin have an asterisk (*) in the password field. This means that these users do not log in to the system, but instead play a role for specific programs.

If the password field is empty, then the user can log in to the system without entering a password. You should always set a password in a multiuser system.

/etc/group

The file **/etc/group** stores group information.

The following is a sample /etc/group file:

Figure 7-5

Each line in the file represents a single group record and contains the group name, a field for the password hash, the GID (group ID) and the members of the group. For example:

```
video:x:33:geeko,tux
```

This is the entry for the group video in /etc/group and has a GID of **33**. Users geeko and tux are members of this group. The **x** in the second field indicates that no password has been set.

The /etc/groups file shows secondary group memberships only; it does not identify the primary group for a user.

In older versions of SUSE LINUX (such as SUSE LINUX Enterprise Server 8), group passwords are stored in the file /etc/gshadow.

Check /etc/passwd and /etc/shadow

Because user configuration is handled by two files (/etc/passwd and /etc/shadow), these files have to match each other. Both files have to contain an entry for each user.

However, discrepancies can occur—especially if you are configuring these files in an editor. There are programs you can use to check for discrepancies in /etc/passwd and /etc/shadow.

For example, to view the contents of both files at once, you can enter the following:

```
da10:~ # tail -3 /etc/passwd /etc/shadow
==> /etc/passwd <==
cyrus:x:96:12:User for cyrus-imapd:/usr/lib/cyrus:/bin/bash
tux:x:1000:100:tux:/home/tux:/bin/bash
geeko:x:1001:100:geeko:/home/geeko:/bin/bash
==> /etc/shadow <==
postfix:!:12543:0:99999:7:::
cyrus:!:12543:0:99999:7:::
tux:0C9zaAMz3p72g:12551:0:99999:7:::
da10:~ #
```

In the above example, the user geeko is entered in /etc/passwd but not in /etc/shadow.

In order to correct this type of error, you can enter the command pwconv:

```
da10:~ # pwconv
da10:~ # tail -3 /etc/passwd /etc/shadow
==> /etc/passwd <==
cyrus:x:96:12:User for cyrus-imapd:/usr/lib/cyrus:/bin/bash
tux:x:1000:100:tux:/home/tux:/bin/bash
geeko:x:1001:100:geeko:/home/geeko:/bin/bash
==> /etc/shadow <==
cyrus:!:12543:0:99999:7:::
tux:0C9zaAMz3p72g:12551:0:99999:7:::
geeko:x:12566:0:99999:7:::0
da10:~ #
```

You can also use the command pwck:

```
da10:~ # pwck
Checking '/etc/passwd'
User 'geeko': directory '/home/geeko' does not exist.
Checking '/etc/shadow'.
da10:~ #
```

Convert Passwords to and from Shadow

To convert passwords to and from /etc/shadow, you can use the **pwconv** command.

This command will also resolve discrepancies where an entry exists in /etc/passwd but not in /etc/shadow.

The following is described:

- Convert Password to Shadow
- Convert Shadow to Password

Convert Password to Shadow

Since /etc/shadow did not exist in early Linux distributions, pwconv was written to help system administrators use the added security that /etc/shadow provides.

The pwconv command converts the passwd file to the shadow file. When you enter pwconv at the command line, it creates /etc/shadow with information from /etc/passwd.

pwconv moves the user password from /etc/passwd to /etc/shadow and replaces the password in /etc/passwd with the special character *x*.

Password aging information (PASS_MIN_DAYS, PASS_MAX_DAYS, PASS_WARN_AGE) is pulled from login.defs and added to /etc/shadow.

If you already have both /etc/passwd and /etc/shadow but the shadow file does not have all the entries that are in the passwd file, pwconv adds the missing entries to the shadow file.

pwconv looks for the special character *x* in /etc/passwd so that it does not modify entries that are already in /etc/shadow.

Convert Shadow to Password

To remove /etc/shadow and convert your user accounts to /etc/passwd only, use the **pwunconv** command. Passwords are moved from /etc/shadow to /etc/passwd and password aging information is lost.

There is no real reason to convert shadow to password. You should avoid doing it, because a separate /etc/shadow is more secure.

Objective 2 Manage User Accounts and Groups from the Command Line

In addition to the YaST modules **users** and **groups**, you can use the following commands to add, change, and delete users and groups:

- useradd
- passwd
- usermod
- userdel
- groupadd, groupmod, and groupdel

To prevent individual users from using system resources too excessively, use the following command:

- ulimit

useradd

You can use the command useradd to add users,. In the simplest case, you enter a user name as an argument, as in the following:

useradd geeko

By entering useradd geeko, the user geeko is created in /etc/passwd and /etc/shadow.

If you don't specify an option, in SLES the command useradd creates a user without a home directory and without a valid password.

The following are the most important options of the command useradd:

- **-m.** This option automatically generates the home directory for the user.

Without further arguments, the directory is created under /home/.

In addition, a series of files and directories are copied to this directory.

The directory /etc/skel/ (from skeleton) is used as a template for the user home directory.

- **-c.** When creating a new user, you can enter text for the comment field by using the option **-c "*comment*"**.
- **-g.** This option defines the primary group of the user.

  You can specify either the GID or the name of the group.
- **-G.** This option defines any supplementary groups (separated by a comma) the user should be a member of.

  You can specify either the GID or the name of the group.
- **-p.** This option lets you create a password for a new user.

  The following is an example:

  useradd -m -p "ghvkuzfFGW6cw" geeko

The encrypted password must be given here, not the plain text password. The program **mkpasswd** can be used to generate encrypted passwords. It is located in the package whois.

- **-e.** The option -e (expiredate) lets you set an expiration date for the user account, in the form of YYYY-MM-DD, as in the following:

  useradd -m -e 2005-09-21 geeko

You can see a description of additional options by entering **man 8 useradd**.

When creating a user account, the necessary standard configuration information (such as primary group, location of the home directory, and default shell) is derived from the files /etc/default/useradd (which is also used by YaST) and /etc/login.defs.

The following is an example of the file /etc/default/useradd:

```
GROUP=1001
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
GROUPS=audio,dialout,uucp,video
```

passwd

You can change a user's password with the command **passwd**.

If a user enters **passwd** without a username as an argument, the user can change her own password.

Besides being able to change a user password, the passwd command provides the following features:

- **Lock a user account.** With the option -l (lock), a user can be kept from logging in. With the option -u (unlock), he can log in again:

```
da10:~ # passwd -l tux
Password changed.
```

- **Show the status of a user account.** The option -S lists the status of a user account:

```
da10:~ # passwd -S tux
tux LK 04/19/2004 0 99999 7 0
```

The status follows directly after the user name. In the above example, LK (locked) means that the user cannot log in. Other options are NP (no password) or PS (valid password).

These are followed by the date of the last password change, the minimum amount of time a password is valid, the maximum amount of time a password is valid, and the warning periods and inactivity periods when a password expires.

- **Change password times.** You can change the password times by using the following options:

Table 7-1

Option	Description
-i *number*	Disables an account after the password has been expired for number of days.
-n *number*	Sets the minimum number of days before a password can be changed.
-w *number*	Warns the user that in number of days her password will expire.
-x *number*	Sets the maximum number of days a password remains valid. After number of days the password must be changed.

The following is an example:

passwd -x 30 -w 5 tux

In this example, the password of the user tux remains valid for 30 days. After this time, the password is required to be changed by tux. Tux receives a warning five days before the password expires.

When the command passwd is used to establish or change the password of a user account, the file /etc/default/passwd is checked for the encryption method to be used:

```
# This file contains some information for
# the passwd (1) command and other tools
# creating or modifying passwords.

 Define default crypt hash
# CRYPT={des,md5,blowfish}
CRYPT=des
...
```

The default setting for the variable CRYPT is DES. Other possible encryption methods include MD5 and Blowfish. YaST also uses the file /etc/default/passwd.

In SLES9, a different algorithm (like blowfish) configured in /etc/security/pam_unix2.conf takes precedence over the one given in /etc/default/passwd.

The default option DES supports only passwords with a length up to eight characters long. MD5 and Blowfish support longer passwords.

The quickest way to create a new user from a command line is to use useradd and passwd, as in the following:

```
da1:~ # useradd -m -c "Geeko Chameleon" geeko
da1:~ # passwd geeko
New password:
Re-enter new password:
Password changed
```

With useradd the user is created, and with passwd the password is entered.

usermod

With **usermod**, you can modify information such as the UID, the standard shell, the home directory, and the primary group in an existing user account.

The usermod options are nearly the same as the options of the command useradd.

The following are some examples of usermod:

- Change the home directory:

  usermod -d /newhome/tux -m tux

- Change the UID:

  usermod -u 1504 tux

userdel

You can use the **userdel** command to delete user accounts.

To remove user accounts from the system, use a command similar to the following:

userdel tux

If you don't specify any options, userdel removes the user from the files

- /etc/passwd
- /etc/shadow
- /etc/group

If the file /var/spool/cron/tabs/*username* exists, it is deleted.

However, the home directory and the data in the home directory is not deleted.

If you want to delete the user's home directory and the data it contains, use the option -r:

userdel -r tux

groupadd, groupmod, and groupdel

You can use the following command line commands to perform the same group management tasks available with YaST (and some tasks not available with YaST):

You need to be logged in as root (or switch to root by entering **su -**) to use these commands.

- **groupadd.** You can create a new group by entering **groupadd *group_name***.

  In this case, the next free GID is used.

  Using the option -g (such as **groupadd -g 200 sports**) lets you specify a GID.

  Using the option -p (such as **groupadd -p SHIXKBmugEhdk sports**) lets you specify a password. You can use the command mkpasswd to create the encrypted password.

  You can verify that the group has been added to the system by entering **tail /etc/group**.

- **groupmod.** You can modify the settings (such as GID, group name, and users) for an existing group.

  The following are examples:

  - Change the GID:

    groupmod -g 201 sports

  - Change the group name from sports to water:

    groupmod -n water sports

- Add the user tux to the group:

  groupmod -A tux sports

- **groupdel.** You can delete a group by entering **groupdel *group_name***. There are no options for this command.

  You can delete a group only if no user has this group assigned as a primary group.

You can learn more about these commands by referring to the manual pages (such as **man groupadd**) or the online help page (such as **groupadd --help**).

Exercise 7-1 Manage User Accounts

In this exercise command line tools are employed to manager user accounts. Especially if there are many accounts to manage, the command line tools usually get the job done faster than YaST.

However, usually you have to use more than one tool, whereas in YaST everything is within one or two dialogs.

To manage user accounts, do the following:

1. Open a terminal window; then su to root (**su -**) with a password of **novell**.
2. Create a new local user by entering

   useradd -c "Tux Linux" -m tux
3. Verify that a home directory for tux was created by entering

   ls /home
4. Verify that there is a entry for the tux user in /etc/shadow by entering

   cat /etc/shadow

   Notice the ! in the second field, indicating that there is no password for tux. You did not use the option –p when creating tux, so no password is set.
5. Add a password for the user tux by entering

   passwd tux
6. Enter the password **suse** twice.
7. Log out as root by entering

   exit
8. Log in as tux by entering

   su - tux
9. Enter the tux password (**suse**).

10. Change the password of the user tux by entering

    passwd

11. Enter the old password of the user tux (**suse**).

12. Try to change the password to novell by entering

    novell

    You see a warning that the password is too simple.

13. Enter **d1g1t@l** as new password (twice).

14. Log out as user tux by entering

    exit

15. Switch to user root (**su -**) with a password of **novell.**

16. Delete the user tux by entering

    userdel -r tux

17. Verify that the home directory for tux has been removed by entering

    ls /home

18. Verify that there is no entry for tux in /etc/passwd by entering

    cat /etc/passwd

19. Close the terminal window.

(End of Exercise)

ulimit

The ulimit command does not have a direct impact on the system performance.

Rather, ulimit prevents individual users from using system resources excessively at the expense of other users.

Accordingly, ulimit can be used to configure

- The memory usage.
- The number of possible processes.
- Other factors.

You can view the current limits by entering

ulimit -a

The output looks like the following:

```
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
file size               (blocks, -f) unlimited
max locked memory       (kbytes, -l) unlimited
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
stack size              (kbytes, -s) unlimited
cpu time               (seconds, -t) unlimited
max user processes              (-u) 1023
virtual memory          (kbytes, -v) unlimited
```

Individual values can also be reset for the current shell and its child processes by using ulimit with one of the options given above, as in the following example:

```
da10:~ # ulimit -u
1023
da10:~ # ulimit -u 100
da10:~ # ulimit -u
100
```

The details of the individual options are described in the manual pages of bash, section ulimit.

You can change the settings globally for the entire system.

The configuration can be performed

- By means of the file /etc/profile.

  or

- By way of the PAM configuration.

  (PAM stands for Pluggable Authentication Modules and is the framework used to perform and configure authentication for various programs requiring authentication, such as login, xdm, or ftp.)

The advantages of using PAM to make configuration changes are:

- The file /etc/security/limits.conf enables user- or group-specific configuration.
- The files in the directory /etc/pam.d/ allow application-specific (login, sshd, etc.) configuration.

The file /etc/profile contains preconfigured entries that you can customize according to your needs, as shown in the following:

```
...
# Adjust some size limits (see bash(1) -> ulimit)
# Note: You may use /etc/initscript instead to set up ulimits and your
PATH.
#
if test "$is" != "ash" ; then
    #ulimit -c 20000      # only core-files less than 20 MB are written
    #ulimit -d 15000      # max data size of a program is 15 MB
    #ulimit -s 15000      # max stack size of a program is 15 MB
    #ulimit -m 30000      # max resident set size is 30 MB

    ulimit -Sc 0          # don't create core files
    ulimit -Sd unlimited
    # ksh does not support this command.
    test "$is" != "ksh" && ulimit -Ss unlimited
    ulimit -Sm unlimited
fi
...
```

The corresponding configuration in the file /etc/security/limits.conf appears as follows:

```
# /etc/security/limits.conf
#
#Each line describes a limit for a user in the form:
#<domain>      <type>  <item>         <value>
...
#*               soft    core            0
#*               hard    rss             10000
#@student        hard    nproc           20
#@faculty        soft    nproc           20
#@faculty        hard    nproc           50
#ftp             hard    nproc           0
#@student        -       maxlogins       4

# Max number of processes for the members
# of the group users
@users           hard    nproc           100
```

This file also contains an explanation of what you can enter in the individual columns.

Exercise 7-2 Use ulimit

The program ulimit is useful when there are several users on a machine and you want to prevent them from giving each other a hard time by using too many of the available resources.

To use ulimit, do the following:

1. Enter the following:

```
tux@da10:~> echo "main() {for(;;)fork();}" > fork.c
tux@da10:~> gcc fork.c
```

The program (a.out) is for demonstration purposes only.

This kind of program is referred to as *fork bomb*.

The program continuously starts new instances of itself, making the computer virtually unusable due the multitude of processes—unless suitable precautions are taken before the program is started.

Do not execute this program on productive systems!

2. Set ulimit to 10.
3. Start a.out.
4. Switch to another console and look at the process table by entering

   ps aux

5. Terminate a.out by pressing **Ctrl + c**.
6. Change the ulimit value.
7. Execute a.out again.
8. Observe the change in the processes.

If the default ulimit value of 1023 is used, the computer will be virtually unusable following the execution of a.out.

Often, the only thing you can do in such a case is to reboot the system.

(End of Exercise)

Objective 3 Manage File Permissions and Ownership

The current file permissions and ownership are displayed using ls -l, as shown in the following example:

```
geeko@da10:~ > ls -la hello.txt
-rw-r--r-- 1 geeko users 0 2004-04-06 12:40 hello.txt
```

The first 10 columns have the following significance:

- **1**. File type (such as -: normal file, d: directory, and l: link).
- **2-4**. File permissions of the user (u) who owns the file (**r**ead, **w**rite, and e**x**ecute).
- **5-7**. File permissions of the owning group (g) of the file (**r**ead, **w**rite, and e**x**ecute).
- **8-10**. File permissions of others (o) (not the owner and not a member of the group) (**r**ead, **w**rite, and e**x**ecute).

For files and directories the significance of the r, w, and x permission is slightly different, as shown in the following table:

Table 7-2

Permission	File	Directory
r	Read the content of the file	List the directory contents
w	Change the content of the file	Create and delete files within the directory
x	Execute the file	Change into the directory

You can change the current values associated with ownership and permissions by knowing how to do the following:

- Change the File Permissions with chmod

- Change the File Ownership with chown and chgrp
- Modify Default Access Permissions
- Configure Special File Permissions

Change the File Permissions with chmod

You can use the command chmod to add (**+**) or remove (**-**) permissions. Both the owner of a file and root can use this command.

The following are examples of using the command chmod:

Table 7-3

Example	Result
chmod u+x	The owner is given permission to execute the file.
chmod g=rw	All group members can read and write to the file.
chmod u=rwx	The owner receives all permissions.
chmod u=rwx,g=rw,o=r	All permissions for the owner, read and write for the group, and read for all other users.
chmod +x	All users (owner, group, others) receive executable permission (depending on umask).
chmod a+x	All users (owner, group, and others) receive executable permission (**a** for **all**).

In the following example, the user geeko allows the other users in the group users to write to the file hello.txt by using chmod:

```
geeko@da10:~ > ls -la hello.txt
-rw-r--r-- 1 geeko users 0 2004-04-06 12:40 hello.txt
geeko@da10:~ > chmod g+w hello.txt
geeko@da10:~ > ls -la hello.txt
-rw-rw-r-- 1 geeko users 0 2004-04-06 12:40 hello.txt
```

With the option -R and a specified directory, you can change the access permissions of all files and subdirectories under the specified directory.

Besides using letters (**rwx**), you can also use the octal way of representing the permission letters with groups of numbers:

Table 7-4

Owner	Group	Others
rwx	rwx	rwx
421	421	421

By using number equivalents, you can add the numbers, as in the following:

Table 7-5

Owner	Group	Others
rwx	rw-	r-x
421 (4+2+1=7)	42- (4+2=6)	4-1 (4+1=5)

The following are examples of using numbers instead of letters:

Table 7-6

Example	Result
chmod 754 hello.txt	All permissions for the owner, read and execute for the group, and read for all other users.

(continued) **Table 7-6**

Example	Result
chmod 777 hello.txt	All users (user, group, and others) receive all permissions.

Depending on what you want to do, either method has its advantages. Which one you prefer is up to you, of course, but you have to know them both.

If you want to add write permissions to the group, no matter the current permission, use **g+w**, like in the following example:

```
da10:/tmp # ls -l hello.txt
-rw-r--r-- 1 geeko users 0 2004-04-06 12:43 hello.txt
da10:/tmp # chmod g+w hello.txt
da10:/tmp # ls -la hello.txt
-rw-rw-r-- 1 geeko users 0 2004-04-06 12:43 hello.txt
```

If you have a certain set of permissions in mind that the file should have, the octal syntax is usually the most efficient.

Suppose you want to achieve rwx for the user, r for the group and no permissions for others. You could figure out what permissions to add and which to subtract, depending on the existing permissions. Another approach would be to use **o=rwx,g=r,o=**, which sets the permissions independently from the existing permissions. However, the octal syntax is much shorter, as the following example shows:

```
da10:/tmp # ls -l hello.sh
-rw-r--r-- 1 geeko users 0 2004-04-06 12:43 hello.txt
da10:/tmp # chmod 740 hello.sh
da10:/tmp # ls -la hello.sh
-rwxr----- 1 geeko users 0 2004-04-06 12:43 hello.txt
```

With a little practice you will get used to the octal syntax, even if it is not intuitive in the beginning.

Change the File Ownership with chown and chgrp

The user root can use the command chown to change the user and group affiliation of a file by using the following syntax:

chown ***new_user.new_group file***

To change only the owner, not the group, use the following command syntax:

chown ***new_user file***

To change only the group, not the user, use the following command syntax:

chown ***.new_group file***

As root, you can also change the group affiliation of a file by using the chgrp command. Use the following syntax:

chgrp ***new_group file***

A normal user can use the command chown to allocate a file that she owns to a new group by using the following syntax:

chown ***.new_group file***

The user can also do the same by using chgrp with the following syntax:

chgrp ***new_group file***

The user can only change the group affiliation of the file that he owns if he is a member of the new group.

In the following example, root changes the ownership of the file hello.txt from geeko to the user newbie by using chown:

```
da10:/tmp # ls -la hello.txt
-rw-r--r-- 1 geeko users 0 2004-04-06 12:43 hello.txt
da10:/tmp # chown newbie.users hello.txt
da10:/tmp # ls -la hello.txt
-rw-r--r-- 1 newbie users 0 2004-04-06 12:43 hello.txt
da10:/tmp #
```

In the following example, chown is used to limit access to the file list.txt to members of the group advanced:

```
da10:/tmp # ls -la list.txt
-rw-r----- 1 geeko users 0 2004-04-06 12:43 list.txt
da10:/tmp # chown .advanced list.txt
da10:/tmp # ls -la list.txt
-rw-r----- 1 geeko advanced 0 2004-04-06 12:43 list.txt
da10:/tmp #
```

User root and the file owner continue to have rights to access the file.

Although the group has changed, the owner permissions remain the same.

Modify Default Access Permissions

If the default settings are not changed, files are created with the access mode **666** and directories with **777** by default.

To modify (restrict) these default access mode settings, you can use the command *umask*. You use this command with a three-digit numerical value such as **022**.

How can you calculate the default setting for file and directory permissions from the umask value? The permissions set in the umask are removed from the default permissions.

The following table shows the permissions assigned to newly created directories and files after setting **umask 022**:

Table 7-7

	Directories	Files
Default Permissions	rwx rwx rwx 7 7 7	rw- rw- rw- 6 6 6
umask	--- -w- -w- 0 2 2	--- -w- -w- 0 2 2
Result	rwx r-x r-x 7 5 5	rw- r-- r-- 6 4 4

The following table shows the permissions assigned to newly created directories and files after setting **umask 023**:

Table 7-8

	Directories	Files
Default Permissions	rwx rwx rwx 7 7 7	rw- rw- rw- 6 6 6
umask	--- -w- -wx 0 2 3	--- -w- -wx 0 2 3
Result	rwx r-x r-- 7 5 4	rw- r-- r-- 6 4 4

In the second example (umask 023), the **x** permission in the umask does not have any effect on the file permissions, as the x permission is missing in the default setting (rw- rw- rw-, 666).

By entering **umask 077** you restrict access to the owner and root only; the group and others do not have any access permissions.

To make the umask setting permanent, you can change the value of umask in the system-wide configuration file /etc/profile.

If you want the setting to be user-specific, enter the value of umask in the file .bashrc in the home directory of the respective user.

Configure Special File Permissions

The following three attributes are used for special circumstances (the uppercase letter is displayed in the output of ls -l in the absence of the execute bit):

Table 7-9

Letter	Number	Name	Files	Directories
t or T	1	Sticky bit	Not applicable	Users can only delete files when they are the owner, or when they are root or owner of the directory. This is usually applied to the directory /tmp/.
s or S	2	SGID (set GroupID)	When a program is run, this sets the group ID of the process to that of the group of the file.	Files created in this directory belong to the group to which the directory belongs and not to the primary group of the user. New directories created in this directory inherit the SGID bit.

(continued) **Table 7-9**

Letter	Number	Name	Files	Directories
s or S	4	SUID (set UserID)	Sets the user ID of the process to that of the owner of the file when the program is run.	Not applicable.

You set the sticky bit with chmod, either via the permissions of others (such as **chmod o+t /tmp**) or numerically (such as **chmod 1777 /tmp**).

The sticky bit on older UNIX systems enabled the storing of an executable program in memory after it had been terminated, so it could be quickly restarted. However, with modern UNIX and Linux systems, this only affects directories.

The sticky bit is listed in the permissions for others (t), as in the following:

```
geeko@da10:~ > ls -ld /tmp
drwxrwxrwt 15 root root 608 2004-04-06 12:45 /tmp
```

The following is an example for SUID:

```
geeko@da10:~ > ls -l /usr/bin/passwd
-rwsr-xr-x 1 root shadow 79765 2004-03-24 12:19 /usr/bin/passwd
```

You can set this bit either by entering

chmod u+s /usr/bin/passwd

or

chmod 4755 /usr/bin/passwd

The following is an example for SGID:

```
geeko@da10:~ > ls -l /usr/bin/wall
-rwxr-sr-x 1 root tty 10192 2004-03-22 05:24 /usr/bin/wall
```

You can set this bit either by entering

chmod g+s /usr/bin/wall

or

chmod 2755 /usr/bin/wall

If the attributes SUID or SGID are set, the programs are carried out with the privileges the owner (in the example for SUID above: root) or the group (in the example for SGID above: tty) have.

If root is the owner of the program, the program is carried out with the permissions of root. Unfortunately, there is a certain security risk in doing this.

For example, it is possible for a user to take advantage of an error in the program, retaining root privileges after the process has been ended.

Exercise 7-3 Manage File Permissions and Ownership

File permissions and ownership is a subject any user on a Linux system needs to understand. For a system administrator this understanding is of crucial importance, as faulty permissions can have serious impact on the system security.

To manage file permissions and ownership, do the following:

1. As user geeko open a terminal window (do not su to root).
2. Create two files:

   echo hello > perm_test1

   echo hello > perm_test2
3. Allow only user geeko to read and write the file perm_test1 by entering

   chmod 600 perm_test1
4. Verify that the change was made by entering

   ls -l

   Notice that geeko is the owner of the file, and that the only permissions assigned to the file are rw for the file owner.

   Also notice that others can read (r) the contents of the file perm_test2.
5. Remove the read permission for others of the file perm_test2 by entering

   chmod o-r perm_test2
6. Make sure that the permissions are correct by entering

   ls -l perm_test*
7. Su to root (**su -**) with a password of **novell**.
8. Create a file df by entering

   touch df

9. Verify that the file was created by entering

   ls -l df

   The owner of the file is root, and the file group is also root.

10. Change the owner of the file df to nobody by entering

    chown nobody df

11. Change the group of the file df to nogroup by entering

    chgrp nogroup df

12. Make sure that the settings are correct by entering

    ls -l df

13. Log out as user root by entering

    exit

14. Close the terminal window.

(End of Exercise)