

Statement of Purpose:

This lab will introduce the basic concept of file access permissions and package management in Linux to you.

Activity Outcomes:

This lab teaches you the following topics:

1. Reading and setting file permissions.
 2. Setting the default file permissions.
 3. Performing package management tasks.
-

1) Stage J (Journey)

1. File Permissions

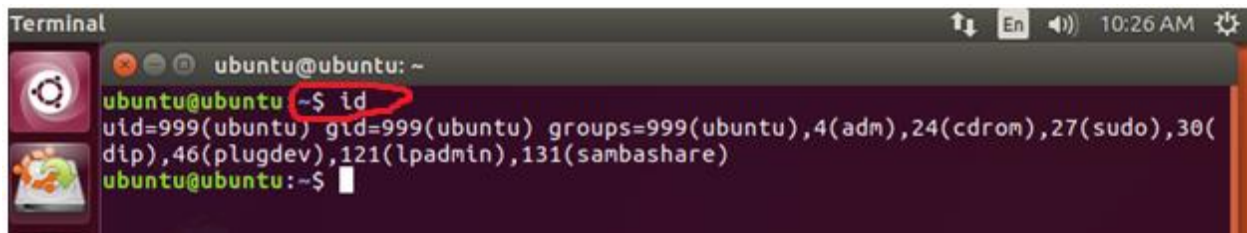
Linux is a multi-user system. It means that more than one person can be using the computer at the same time. While a typical computer will likely have only one keyboard and monitor, it can still be used by more than one user. For example, if a computer is attached to a network or the Internet, remote users can log in via ssh (secure shell) and operate the computer. In fact, remote users can execute graphical applications and have the graphical output appear on a remote display.

In a multi-user environment, to ensure the operational accuracy, it is required to protect the users from each other. After all, the actions of one user could not be allowed to crash the computer, nor could one user interfere with the files belonging to another user.

1.1 id command

In the Linux security model, a user may own files and directories. When a user owns a file or directory, the user has control over its access. Users can, in turn, belong to a group consisting of one or more users who are given access to files and directories by their owners. In addition to granting access to a group, an owner may also grant some set of access rights to everybody, which in Linux terms is referred to as the world.

User accounts are defined in the `/etc/passwd` file and groups are defined in the `/etc/group` file. When user accounts and groups are created, these files are modified along with `/etc/shadow` which holds information about the user's password.

A terminal window titled 'Terminal' with a dark background. The prompt is 'ubuntu@ubuntu: ~'. The command 'id' has been entered and executed. The output is: 'uid=999(ubuntu) gid=999(ubuntu) groups=999(ubuntu),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),121(lpadmin),131(sambashare)'. The command 'id' is circled in red. The prompt returns to 'ubuntu@ubuntu:~\$'.

Option	Explanation
-g	Print only the effective group id
-G	Print all Group ID's
-n	Prints name instead of number.
-r	Prints real ID instead of numbers.

-u	Prints only the effective user ID.
----	------------------------------------

1.2 Reading, Writing, and Executing

Access rights to files and directories are defined in terms of read access, write access, and execution access. If we look at the output of the ls command, we can get some clue as to how this is implemented:

```

Terminal
ubuntu@ubuntu: ~
ubuntu@ubuntu:~$ ls -l
total 0
drwxr-xr-x 2 ubuntu ubuntu 80 Sep 16 10:22 Desktop
drwxr-xr-x 2 ubuntu ubuntu 40 Sep 16 10:23 Documents
drwxr-xr-x 2 ubuntu ubuntu 40 Sep 16 10:23 Downloads
drwxr-xr-x 2 ubuntu ubuntu 40 Sep 16 10:23 Music
drwxr-xr-x 2 ubuntu ubuntu 40 Sep 16 10:23 Pictures
drwxr-xr-x 2 ubuntu ubuntu 40 Sep 16 10:23 Public
drwxr-xr-x 2 ubuntu ubuntu 40 Sep 16 10:23 Templates
drwxr-xr-x 2 ubuntu ubuntu 40 Sep 16 10:23 Videos
ubuntu@ubuntu:~$

```

The first ten characters of the listing are the file attributes. The first of these characters is the file type. Here are the file types you are most likely to see:

Attribute	File Type
-	A regular file.
d	A directory.
l	A symbolic link.
c	A character special file. This file type refers to a device that handles data as a stream of bytes, such as a terminal or modem.
b	A block special file. This file type refers to a device that handles data in blocks, such as a hard drive or CD-ROM drive.

The remaining nine characters of the file attributes, called the file mode, represent the read, write, and execute permissions for the file's owner, the file's group owner, and everybody else.

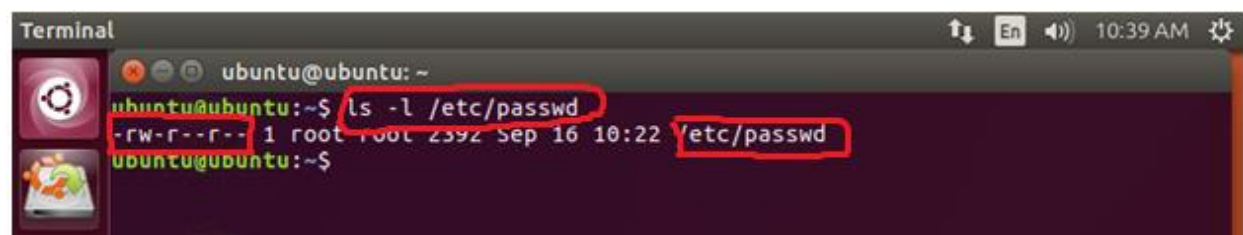
User	Group	World
rwX	rwX	rwX

Attribute	Files	Directories
r	Allows a file to be opened and read.	Allows a directory's contents to be listed if the execute attribute is also set.
w	Allows a file to be written	Allows files within a directory to be created, deleted, and renamed if the execute attribute is also set.
x	Allows a file to be treated as a program and executed.	Allows a directory to be entered, e.g., cd directory.

For example: **-rw-r--r-** A regular file that is readable and writable by the file's owner. Members of the file's owner group may read the file. The file is world-readable.

1.3 Reading File Permissions

The `ls` command is used to read the permission of a file. In the following example, we have used `ls` command with `-l` option to see the information about `/etc/passwd` file. Similarly, we can read the current permissions of any file.



```

Terminal
ubuntu@ubuntu: ~
ubuntu@ubuntu:~$ ls -l /etc/passwd
-rw-r--r-- 1 root root 2392 Sep 16 10:22 /etc/passwd
ubuntu@ubuntu:~$

```

1.4 Change File Mode (Permissions)

To change the mode (permissions) of a file or directory, the **chmod** command is used. Beware that only the file's owner or the super-user can change the mode of a file or directory. **chmod** supports two distinct ways of specifying mode changes: octal number representation, or symbolic representation. With octal notation we use octal numbers to set the pattern of desired permissions. Since each digit in an octal number represents three binary digits, these maps nicely to the scheme used to store the file mode.

Octal	Binary	File Mode
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwX

In the following example, we first go to the Desktop directory using `cd` command. In Desktop directory, we create a text file **“myfile.txt”** using `touch` command and read its current permissions using `ls` command.

```
Terminal
ubuntu@ubuntu: ~/Desktop
ubuntu@ubuntu:~$ cd Desktop
ubuntu@ubuntu:~/Desktop$ touch myfile.txt
ubuntu@ubuntu:~/Desktop$ ls -l
total 20
-rw-r--r-- 1 ubuntu ubuntu 8980 Sep 16 10:22 examples.desktop
-rw-r--r-- 1 ubuntu ubuntu 0 Sep 16 10:40 myfile.txt
-rwxr-xr-x 1 ubuntu ubuntu 7861 Sep 16 10:22 ubiquity.desktop
ubuntu@ubuntu:~/Desktop$
```

Now, we change the permission of `myfile.txt` and set it to 777 that is everyone can read, write and execute the file.

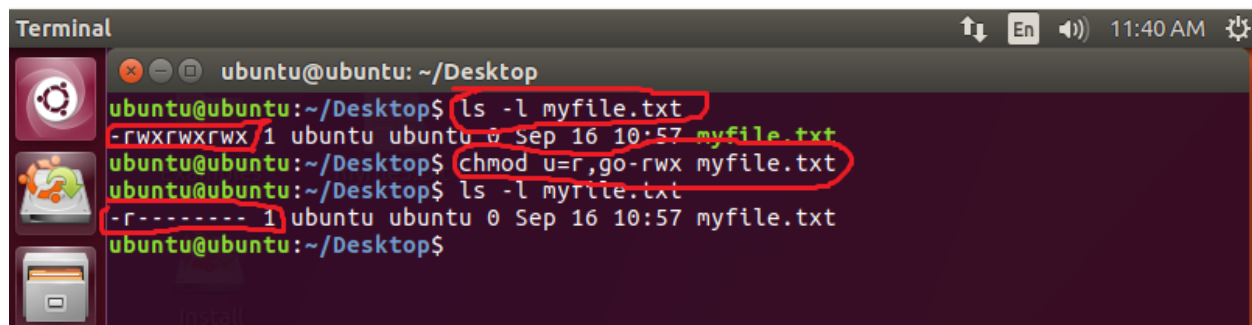
```
Terminal
ubuntu@ubuntu: ~/Desktop
ubuntu@ubuntu:~/Desktop$ chmod 777 myfile.txt
ubuntu@ubuntu:~/Desktop$ ls -l
total 20
-rw-r--r-- 1 ubuntu ubuntu 8980 Sep 16 10:22 examples.desktop
-rwxrwxrwx 1 ubuntu ubuntu 0 Sep 16 10:40 myfile.txt
-rwxr-xr-x 1 ubuntu ubuntu 7861 Sep 16 10:22 ubiquity.desktop
ubuntu@ubuntu:~/Desktop$
```

chmod also supports a symbolic notation for specifying file modes. Symbolic notation is divided into three parts: who the change will affect, which operation will be performed, and what permission will be set. To specify who is affected, a combination of the characters “u”, “g”, “o”, and “a” is used as follows:

Character	Meaning
u	Owner
g	Group
o	Others
a	all

If no character is specified, “all” will be assumed. The operation may be a “+” indicating that a permission is to be added, a “-” indicating that a permission is to be taken away, or a “=” indicating that only the specified permissions are to be applied and that all others are to be removed. For example: **u+x,go=rx** Add execute permission for the owner and set the permissions for the group and others to read and execute. Multiple specifications may be separated by commas.

In the following example, we change the permissions of myfile.txt using symbolic codes. As all of the permissions of myfile.txt were set previously, now we make it readable only to the user while the rest cannot read, write or execute the file.

A terminal window titled "Terminal" with a dark background. The prompt is "ubuntu@ubuntu: ~/Desktop". The user enters "ls -l myfile.txt" and the output is "-rwxrwxrwx 1 ubuntu ubuntu 0 Sep 16 10:57 myfile.txt". Then the user enters "chmod u=r,go-rwx myfile.txt". Finally, the user enters "ls -l myfile.txt" and the output is "-r----- 1 ubuntu ubuntu 0 Sep 16 10:57 myfile.txt". Red circles highlight the command and the resulting permissions in the output.

```
Terminal
ubuntu@ubuntu: ~/Desktop
ubuntu@ubuntu:~/Desktop$ ls -l myfile.txt
-rwxrwxrwx 1 ubuntu ubuntu 0 Sep 16 10:57 myfile.txt
ubuntu@ubuntu:~/Desktop$ chmod u=r,go-rwx myfile.txt
ubuntu@ubuntu:~/Desktop$ ls -l myfile.txt
-r----- 1 ubuntu ubuntu 0 Sep 16 10:57 myfile.txt
ubuntu@ubuntu:~/Desktop$
```

1.5 Controlling the Default Permissions

On Unix-like operating systems, the **umask** command returns, or sets, the value of the system's file mode creation mask. When user create a file or directory under Linux or UNIX, he/she creates it with a default set of permissions. In most case the system defaults may be open or relaxed for file sharing purpose.

umask command with no arguments can be used to return the current mask value. Similarly, If the **umask** command is invoked with **an octal argument**, it will directly set the bits of the mask to that argument.

The three rightmost octal digits address the "owner", "group" and "other" user classes respectively. If fewer than 4 digits are entered, leading zeros are assumed. An error will result if the argument is not a valid octal number or if it has more than 4 digits. If a fourth digit is present, the leftmost (high-order) digit addresses three additional attributes, the setuid bit, the setgid bit and the sticky bit.

Octal Value	Permissions
0	read, write, execute
1	read and write
2	read and execute
3	read only
4	write and execute
5	write only
6	execute only
7	no permissions

In the following example, we first read the current mask that is 0022 and then we created a file myfile.txt using this mask and display its permission. Then we reset the mask with 111 and 333 octal values and create new files. It can be seen clearly that new files are created with different default permissions.

```

Terminal
ubuntu@ubuntu: ~/Desktop
ubuntu@ubuntu:~/Desktop$ umask
0022
ubuntu@ubuntu:~/Desktop$ touch myfile.txt
ubuntu@ubuntu:~/Desktop$ ls -l myfile.txt
-rwxrwxrwx 1 ubuntu ubuntu 0 Sep 16 10:57 myfile.txt
ubuntu@ubuntu:~/Desktop$ umask 111
ubuntu@ubuntu:~/Desktop$ touch myfile2.txt
ubuntu@ubuntu:~/Desktop$ ls -l myfile2.txt
-rw-rw-rw- 1 ubuntu ubuntu 0 Sep 16 11:05 myfile2.txt
ubuntu@ubuntu:~/Desktop$ umask 333
ubuntu@ubuntu:~/Desktop$ touch myfile3.txt
ubuntu@ubuntu:~/Desktop$ ls -l myfile3.txt
-r--r--r-- 1 ubuntu ubuntu 0 Sep 16 11:06 myfile3.txt
ubuntu@ubuntu:~/Desktop$

```

1.6 Changing User Identity

At various times, we may find it necessary to take on the identity of another user. Often, we want to gain superuser privileges to carry out some administrative task, but it is also possible to “become” another regular user for such things as testing an account.

Run A Shell with Substitute User and Group IDs

The su command is used to start a shell as another user. The command syntax looks like

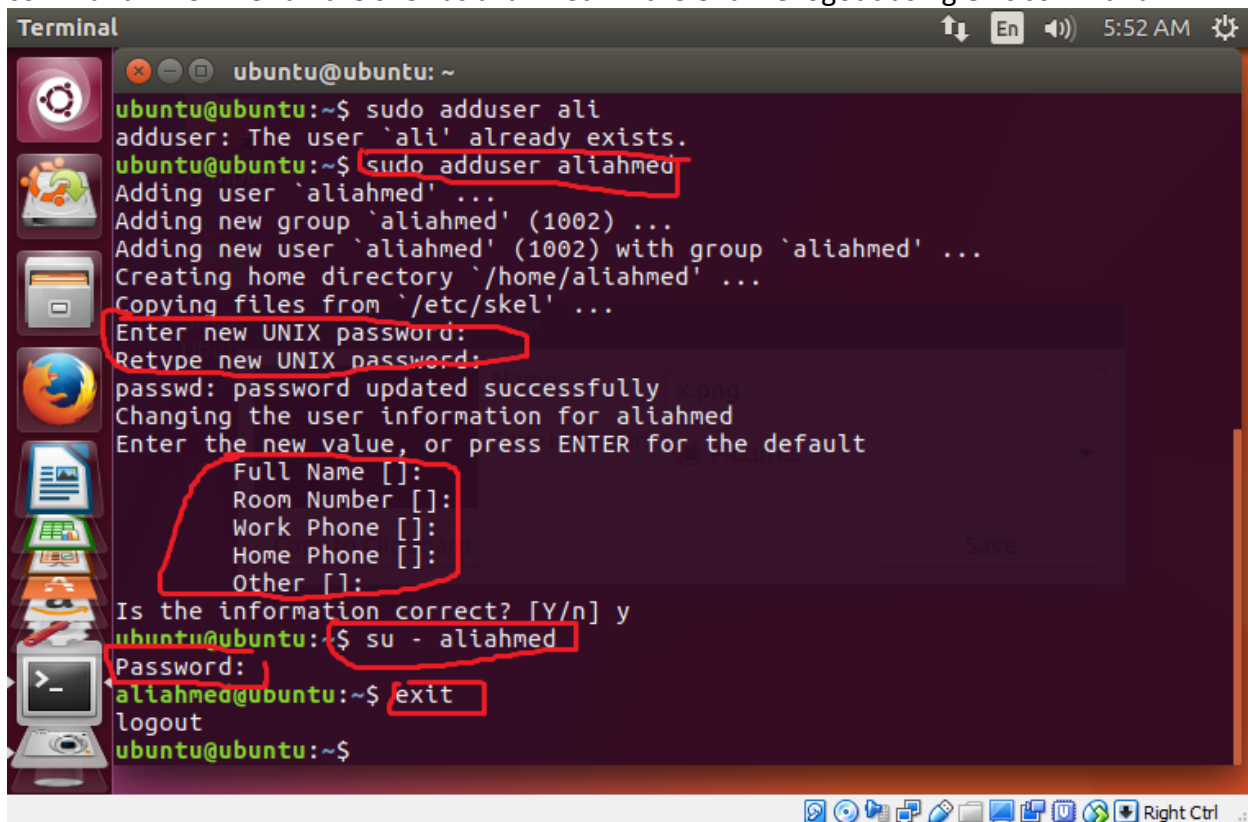
this:

su -l username

Execute A Command as Another User

On Unix-like operating systems, the **sudo** command ("superuser do", or "switch user, do") allows a user with proper permissions to execute a command as another user, such as the superuser.

Example: In the following example first, we created a new user "aliahmed" using **adduser** command. Then we run the shell as aliahmed. In the end we logout using **exit** command.

A terminal window titled 'Terminal' with a dark background and light text. The window shows the process of adding a new user 'aliahmed' using the 'sudo adduser' command. The output shows that the user 'ali' already exists, so 'aliahmed' is added instead. The user is prompted to enter a password, which is then confirmed. The user is then prompted to enter their full name, room number, work phone, home phone, and other information. The user confirms the information is correct. Finally, the user switches to the 'aliahmed' user using 'su - aliahmed' and then logs out using 'exit'.

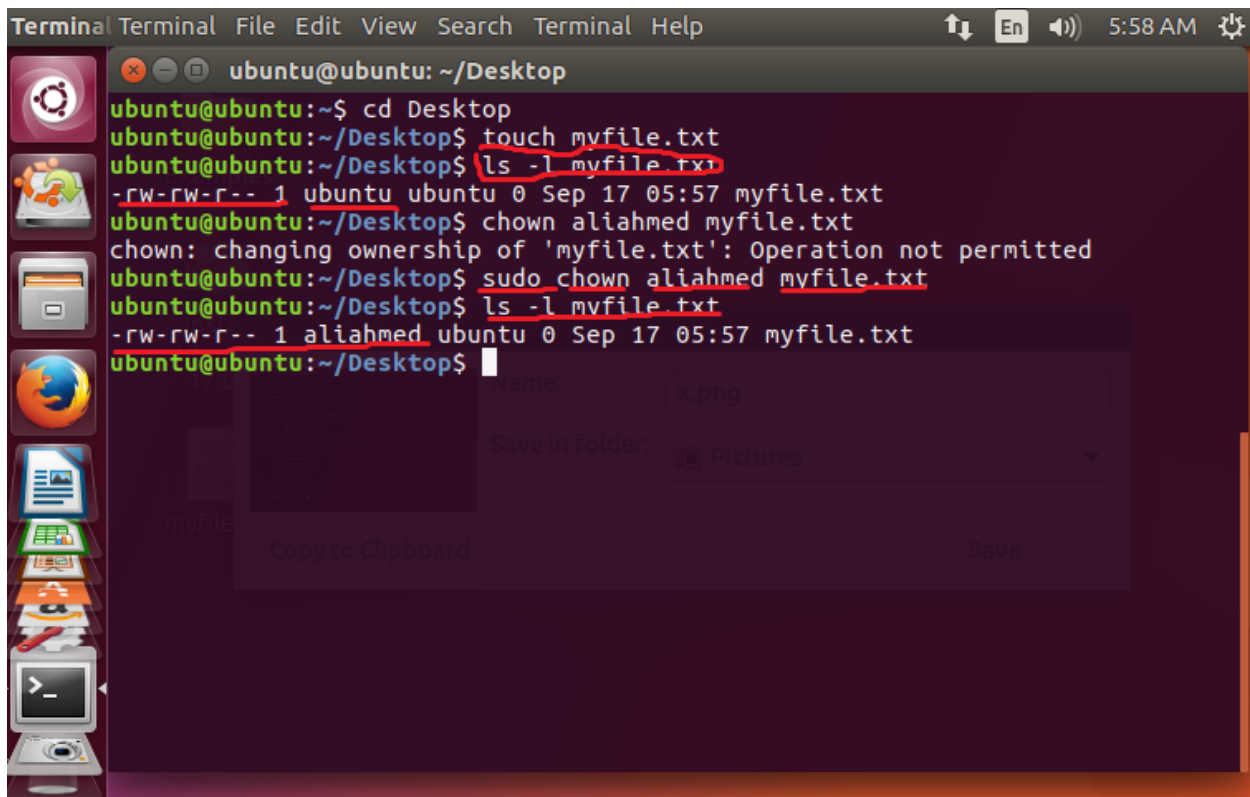
```
Terminal
ubuntu@ubuntu: ~
ubuntu@ubuntu:~$ sudo adduser ali
adduser: The user `ali' already exists.
ubuntu@ubuntu:~$ sudo adduser aliahmed
Adding user `aliahmed' ...
Adding new group `aliahmed' (1002) ...
Adding new user `aliahmed' (1002) with group `aliahmed' ...
Creating home directory `/home/aliahmed' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for aliahmed
Enter the new value, or press ENTER for the default
Full Name []:
Room Number []:
Work Phone []:
Home Phone []:
Other []:
Is the information correct? [Y/n] y
ubuntu@ubuntu:~$ su - aliahmed
Password:
aliahmed@ubuntu:~$ exit
logout
ubuntu@ubuntu:~$
```

1.7 Change File Owner and Group

The **chown** command is used to change the owner and group owner of a file or directory. Superuser privileges are required to use this command. The syntax of **chown** looks like this:

chown owner:group file/files

Example: In the following example first, we created a file named **myfile.txt** as user **ubuntu**. Then we changed the ownership of **myfile.txt** from **ubuntu** to **aliahmed**.



```
Terminal Terminal File Edit View Search Terminal Help
ubuntu@ubuntu: ~/Desktop
ubuntu@ubuntu:~$ cd Desktop
ubuntu@ubuntu:~/Desktop$ touch myfile.txt
ubuntu@ubuntu:~/Desktop$ ls -l myfile.txt
-rw-rw-r-- 1 ubuntu ubuntu 0 Sep 17 05:57 myfile.txt
ubuntu@ubuntu:~/Desktop$ chown aliahmed myfile.txt
chown: changing ownership of 'myfile.txt': Operation not permitted
ubuntu@ubuntu:~/Desktop$ sudo chown aliahmed myfile.txt
ubuntu@ubuntu:~/Desktop$ ls -l myfile.txt
-rw-rw-r-- 1 aliahmed ubuntu 0 Sep 17 05:57 myfile.txt
ubuntu@ubuntu:~/Desktop$
```

A file dialog box is overlaid on the terminal, showing 'Name: x.png' and 'Save in folder: Pictures'. It has 'Copy to Clipboard' and 'Save' buttons.

2. Package Management

Package management is a method of installing and maintaining software on the system. Linux doesn't work that way. Virtually all software for a Linux system will be found on the Internet. Most of it will be provided by the distribution vendor in the form of *package files* and the rest will be available in source code form that can be installed manually.

Different distributions use different packaging systems and as a general rule, a package intended for one distribution is not compatible with another distribution. Most distributions fall into one of two camps of packaging technologies: the Debian “.deb” camp and the Red Hat “.rpm” camp. There are some important exceptions such as Gentoo, Slackware, and Foresight, but most others use one of these two basic systems.

Package Files

The basic unit of software in a packaging system is the package file. A package file is a compressed collection of files that comprise the software package. A package may consist of numerous programs and data files that support the programs. In addition to the files to be installed, the package file also includes metadata about the package, such as a text description of the package

and its contents. Additionally, many packages contain pre- and post-installation scripts that perform configuration tasks before and after the package installation.

Repositories

While some software projects choose to perform their own packaging and distribution, most packages today are created by the distribution vendors and interested third parties. Packages are made available to the users of a distribution in central repositories that may contain many thousands of packages, each specially built and maintained for the distribution.

Dependencies

Programs seldom “standalone”; rather they rely on the presence of other software components to get their work done. Common activities, such as input/output for example, are handled by routines shared by many programs. These routines are stored in what are called *shared libraries*, which provide essential services to more than one program. If a package requires a shared resource such as a shared library, it is said to have a *dependency*. Modern package management systems all provide some method of *dependency resolution* to ensure that when a package is installed, all of its dependencies are installed too

High and Low-level Package Tools

Package management systems usually consist of two types of tools: low-level tools which handle tasks such as installing and removing package files, and high-level tools that perform metadata searching and dependency resolution. For Debian based systems low-level tools are defined in dpkg while high-level tools are defined in apt-get, aptitude.

Common Package Management Tasks

2.1 Finding a Package in a Repository

Using the high-level tools to search repository metadata, a package can be located based on its name or description. In Debian based systems it can be done as given below:

apt-get update

apt-cache search *search_string*

Example: To search apt repository for the emacs text editor, this command could be used:

apt-get update

apt-get search emacs

2.2 Installing a Package from a Repository

High-level tools permit a package to be downloaded from a repository and installed with full dependency resolution.

Example: To install the emacs text editor from an apt repository:

```
apt-get update; apt-get install emacs
```

2.3 Installing a Package from a Package File

If a package file has been downloaded from a source other than a repository, it can be installed directly (though without dependency resolution) using a low-level tool.

```
dpkg-install package_file
```

Example: If the emacs-22.1-7.fc7-i386.deb package file had been downloaded from a non-repository site, it would be installed this way:

```
dpkg --install emacs-22.1-7.fc7-i386.deb
```

2.4 Removing A Package

Packages can be uninstalled using either the high-level or low-level tools. The high-level tools are shown below.

```
apt -get remove package_name
```

Example: To uninstall the emacs package from a Debian-style system:

```
apt -get remove emacs
```

2.5 Updating Packages from a Repository

The most common package management task is keeping the system up-to-date with the latest packages. The high-level tools can perform this vital task in one single step.

Example: To apply any available updates to the installed packages on a Debian-style system:

```
apt -get update; apt-get upgrade
```

2.6 Upgrading a Package from a Package File

If an updated version of a package has been downloaded from a non-repository source, it can be installed, replacing the previous version:

```
dpkg --install package_file
```

2.7 Listing Installed Packages

These commands can be used to display a list of all the packages installed on the system:

`dpkg --list`

2.8 Determining If A Package Is Installed

These low-level tools can be used to display whether a specified package is installed:

`dpkg --status package_name`

Example:

`dpkg --status emacs`

2.9 Displaying Information About an Installed Package

If the name of an installed package is known, the following commands can be used to display a description of the package:

`apt -cache show package_name`

Example:

`apt -cache show emacs`

2) Stage a1 (apply)

Lab Activities

Activity 1:

This activity is related to file permission. Perform the following tasks

1. Create a new directory named test in root directory as superuser
2. Make this directory public for all
3. Create a file "testfile.txt" in /test directory
4. Change its permissions that no boy can write the file, but the owner can read it.
5. Create another user "Usama"
6. Run the shell with user Usama
7. Try to read the "testfile.txt"
8. Logout as Usama
9. Change the permission of testfile.txt so that everyone can read, write and execute it
10. Run shell as Usama again
11. Now, read the file

Activity 2:

Perform the following tasks

1. Set the permission such that a newly created file is readable only to the owner
2. Create a text file "act.txt" in /test directory created in previous activity
3. Run the shell as user "usama" (created previously)
4. Access the file "act.txt"
5. Logout as usama
6. Now change the ownership of the file from ubuntu to usama
7. Run the shell again as usama
8. Read the file "act.txt" using cat command
9. Logout as usama
10. Now access the act.txt with user ubuntu

Activity 3:

Perform the following tasks

1. search apt repository for the chromium browser
2. install the chromium browser using command line
3. write the command to update and upgrade the repository
4. list the software installed on your machine and write output on a file list.txt
5. read the list.txt file using cat command

3) Stage v (verify)

Home Activities:

familiar with adduser command using: `man adduser/useradd`, `man groupadd useradd` - create a new user or update default new user information. Create 3 user accounts (user1, user2, user3) and add 2 groups (gr1, gr2). add user1 to gr1 and add user2, user2 to gr2. Check user ID (UID) and group ID (GID) by listing file `/etc/passwd`. Find lines with added user. What is the UID and GID for these accounts? Write command which show UID and GID for your user name:

2. create 3 files with touch command: file1, file2, file3.
3. Write the command line by using letters with chmod to set the following permissions:

- `rw-rw-r-x` for file1
- `r-x--x--x` for file2
- `--x-rwx` for file3

4. Write the command line by using numbers with chmod to set the following permissions:

- `rw-rw-rwx` for file4 (you have to prepare this file)
- `-w-----` for file5 (you have to prepare this file)
- `rw-x--x--x` for folder1 (you have to prepare this folder)

5. Create two user accounts: tst1 and tst2 Logging in id: tst1, group users, with bash shell, home directory `/home/tst1` Logging in id: tst2, group public, with bash shell, home directory `home/tst2` For the two accounts set a password.

Logging in as tst1 and copy `/bin/ls` into tst1 home directory as myls file. Change the owner of myls to tst1 and the permissions to 0710. What does this permission value mean?

Logging in as tst2 and try to use `/home/tst1/myls` to list your current directory. Does it work ?

Create a new group labo with tst1 and tst2. Change the owner group of myls file to labo. Try again from tst2 account to execute `/home/tst1/myls` to list your current directory. Does it work?