APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY (APJ KTU)

RAJIV GANDHI INSTITUTE OF TECHNOLOGY, KOTTYAM

DEPARTMENT OF COMPUTER APPLICATIONS

20MCA136 NETWORKING & SYSTEM ADMINISTRATION LAB

LAB MANUAL

S2 MCA

L-T-P-Credits:0-1-3-2

Table of Contents

Course Objectives

This laboratory course is intended to provide the background knowledge required for a software professional in the fields of networking and system administration. Students will acquire necessary knowledge to deploy and administer systems.

Course Outcomes:

After the completion of the course the student will be able to

1. Install and configure common operating systems.
2. Perform system administration tasks.
3. Install and manage servers for web applications.
4. Write shell scripts required for system administration.
5. Acquire skill sets required for a DevOps.

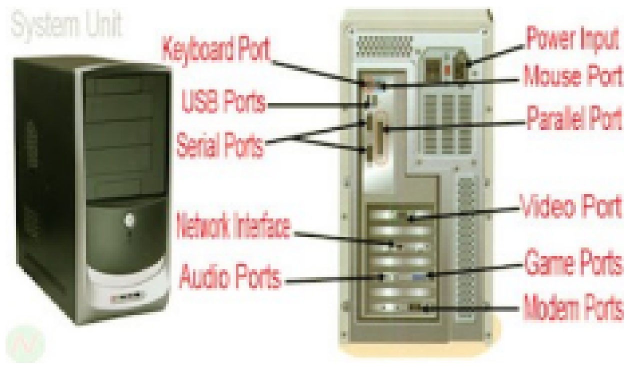
Chapter 1

Introduction To Computer Hardware

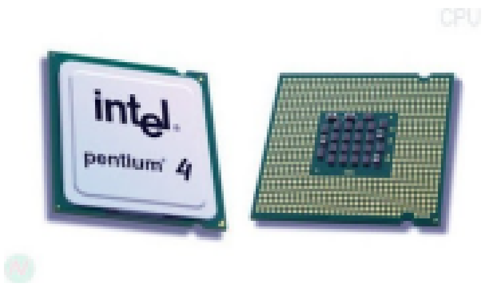
This is intended to introduce students to current hardware available in market a)

Motherboard

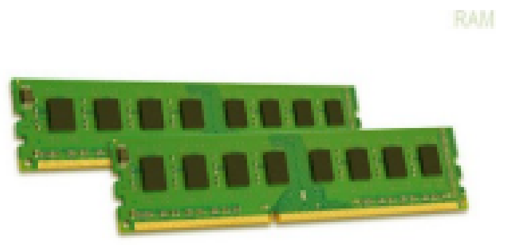
b) SMPS and Cabinet:



c) Processor



d)Memory:



e) Hard disk drives: e) Daughter

cards

f) External Ports VGA,HDMI USB : broad idea about capabilities of external ports

g) Monitors

h) Familiarization of peripheral devices such as scanners and printers

Chapter 2

Exploring Linux File System

It makes sense to explore the Linux filesystem from a terminal window,
In fact, that is the name of the first tool you'll install to help you on the way: *tree*. If you are using Ubuntu or Debian, you can do:

```
sudo apt install tree
```

Once installed, stay in your terminal window and run *tree* like this:

```
$ tree /
```

The / in the instruction above refers to the *root* directory. The root directory is the one from which all other directories branch off from. When you run tree and tell it to start with */*, you will see the whole directory tree, all directories and all the subdirectories in the whole system, with all their files, fly by.
If you have been using your system for some time, this may take a while, because, even if you haven't generated many files yourself, a Linux system and its apps are always logging, caching, and storing temporary files. The number of entries in the file system can grow quite quickly. Instead, try this:

```
tree -L 1 /
```

And you should see a listing similar to what is shown in Figure 1.
Figure 1: tree
The instruction above can be translated as "*show me only the 1st Level of the directory tree*

starting at / (root)". The *-L* option tells *tree* how many levels down you want to see. Most Linux distributions will show you the same or a very similar layout to what you can see in the image above. This means that even if you feel confused now, master this, and you will have a handle on most, if not all, Linux installations in the whole wide world.
Now, let's look at what each directory is used for. While we go through each, you can peek at their contents using *ls*.

Directories

From top to bottom, the directories you are seeing are as follows.

/bin

/bin is the directory that contains *bin*aries, that is, some of the applications and programs you can run. You will find the *ls* program mentioned above in this directory, as well as other basic tools for making and removing files and directories, moving them around, and so on. There are more *bin* directories in other parts of the file system tree, but we'll be talking about those in a minute.

/boot

The */boot* directory contains files required for starting your system. If you mess up one of the files in here, you may not be able to run your Linux and it is a pain to repair. On the other hand, don't worry too much about destroying your system by accident: you have to have superuser privileges to do that.

/dev

/dev contains *dev*ice files. Many of these are generated at boot time or even on the fly. For

example, if you plug in a new webcam or a USB pendrive into your machine, a new device entry will automatically pop up here.

/etc

/etc is the directory where names start to get confusing. */etc* gets its name from the earliest Unixes and it was literally "et cetera" because it was the dumping ground for system files administrators were not sure where else to put.
Nowadays, it would be more appropriate to say that *etc* stands for "Everything to configure," as it contains most, if not all system-wide configuration files. For example, the files that contain the name of your system, the users and their passwords, the names of machines on your network and when and where the partitions on your hard disks should be mounted are all in here. Again, if you are new to Linux, it may be best if you don't touch too much in here until you have a better understanding of how things work.

/home

/home is where you will find your users' personal directories. In my case, under */home* there are two directories: */home/paul*, which contains all my stuff; and */home/guest*, in case anybody needs to borrow my computer.

/lib

/lib is where *lib*raries live. Libraries are files containing code that your applications can use. They contain snippets of code that applications use to draw windows on your desktop, control peripherals, or send files to your hard disk.
There are more *lib* directories scattered around the file system, but this one, the one hanging directly off of */* is special in that, among other things, it contains the all-important kernel modules. The kernel modules are drivers that make things like your video card, sound card, WiFi, printer, and so on, work.

/media

The */media* directory is where external storage will be automatically mounted when you plug it in and try to access it. As opposed to most of the other items on this list, */media* does not hail back to 1970s, mainly because inserting and detecting storage (pendrives, USB hard disks, SD cards, external SSDs, etc) on the fly, while a computer is running, is a relatively new thing. ***/mnt***
The */mnt* directory, however, is a bit of remnant from days gone by. This is where you would manually mount storage devices or partitions. It is not used very often nowadays.

/opt

The */opt* directory is often where software you compile (that is, you build yourself from source code and do not install from your distribution repositories) sometimes lands. Applications will end up in the */opt/bin* directory and libraries in the */opt/lib* directory.
A slight digression: another place where applications and libraries end up in is */usr/local*, When software gets installed here, there will also be */usr/local/bin* and */usr/local/lib* directories. What determines which software goes where is how the developers have configured the files that control the compilation and installation process.

/proc

/proc, like */dev* is a virtual directory. It contains information about your computer, such as information about your CPU and the kernel your Linux system is running. As with */dev*, the files and directories are generated when your computer starts, or on the fly, as your system is running and things change.

/root

/root is the home directory of the superuser (also known as the “Administrator”) of the system. It is separate from the rest of the users’ home directories BECAUSE YOU ARE NOT MEANT TO

TOUCH IT. Keep your own stuff in you own directories, people.

/run

/run is another new directory. System processes use it to store temporary data for their own nefarious reasons.

/sbin

/sbin is similar to */bin*, but it contains applications that only the superuser (hence the initial *s*) will need. You can use these applications with the sudo command that temporarily concedes you superuser powers on many distributions. */sbin* typically contains tools that can install stuff, delete stuff and format stuff. As you can imagine, some of these instructions are lethal if you use them improperly, so handle with care.

/usr

The */usr* directory was where users’ home directories were originally kept back in the early days of UNIX. However, now */home* is where users kept their stuff as we saw above. These days, */usr* contains a mish-mash of directories which in turn contain applications, libraries, documentation,
wallpapers, icons and a long list of other stuff that need to be shared by applications and services.
You will also find *bin*, *sbin* and *lib* directories in */usr*. What is the difference with their root-hanging cousins? Not much nowadays. Originally, the */bin* directory (hanging off of root) would contain very basic commands, like ls, mv and rm; the kind of commands that would come pre-installed in all UNIX/Linux installations, the bare minimum to run and maintain a system. */usr/bin* on the other hand would contain stuff the users would install and run to use the system as a work station, things like word processors, web browsers, and other apps. But many modern Linux distributions just put everything into */usr/bin* and have */bin* point to

/usr/bin just in case erasing it completely would break something. So, while Debian, Ubuntu and Mint still keep */bin* and */usr/bin* (and */sbin* and */usr/sbin*) separate; others, like Arch and its derivatives just have one "real" directory for binaries, */usr/bin*, and the rest or **bin*s are "fake" directories that point to */usr/bin*.

/srv

The */srv* directory contains data for servers. If you are running a web server from your Linux box, your HTML files for your sites would go into */srv/http* (or */srv/www*). If you were running an FTP server, your files would go into */srv/ftp*.

/sys

/sys is another virtual directory like */proc* and */dev* and also contains information from devices connected to your computer.

In some cases you can also manipulate those devices. I can, for example, change the brightness of the screen of my laptop by modifying the value stored in the */sys/devices/pci0000:00/0000:00:02.0/drm/card1/card1-eDP-1/intel_backlight/brightness* file (on your machine you will probably have a different file). But to do that you have to become superuser. The reason for that is, as with so many other virtual directories, messing with the contents and files in */sys* can be dangerous and you can trash your system. DO NOT TOUCH until you are sure you know what you are doing.

/tmp

/tmp contains temporary files, usually placed there by applications that you are running. The files and directories often (not always) contain data that an application doesn't need right now, but may need later on.

You can also use */tmp* to store your own temporary files — */tmp* is one of the few directories hanging off / that you can actually interact with without becoming superuser.

/var

/var was originally given its name because its contents was deemed *variable*, in that it changed frequently. Today it is a bit of a misnomer because there are many other directories that also contain data that changes frequently, especially the virtual directories we saw above. Be that as it may, */var* contains things like logs in the */var/log* subdirectories. Logs are files that register events that happen on the system. If something fails in the kernel, it will be logged in a file in */var/log*; if someone tries to break into your computer from outside, your firewall will also log the attempt here. It also contains *spools* for tasks. These "tasks" can be the jobs you send to a shared printer when you have to wait because another user is printing a long document, or mail that is waiting to be delivered to users on the system.

Your system may have some more directories we haven't mentioned above. In the screenshot, for example, there is a */snap* directory. That's because the shot was captured on an Ubuntu system. Ubuntu has recently incorporated snap packages as a way of distributing software. The */snap* directory contains all the files and the software installed from snaps.

/var/log/syslog contains lot of system related logfiles.

That is the root directory covered, but many of the subdirectories lead to their own set of files and subdirectories. Figure 2 gives you an overall idea of what the basic file system tree looks like (the image is kindly supplied under a CC By-SA license by Paul Gardner) and Wikipedia has a break down with a summary of what each directory is used for.

Figure 2: Standard Unix filesystem hierarchy.

To explore the filesystem yourself, use the cd command:

```
cd
```

will take you to the directory of your choice (*cd* stands for *change*

directory. If you get confused,
pwd
will always tell you where you (*pwd* stands for *print working directory*).
Also, cd
with no options or parameters, will take you back to your own home directory, where things are safe and cosy.
Finally,
cd ..
will take you up one level, getting you one level closer to the */* root directory. If you are in */usr/share/wallpapers* and run cd .., you will move up to */usr/share*.
To see what a directory contains, use
ls
or simply
ls
to list the contents of the directory you are in right now.
And, of course, you always have tree to get an overview of what lays within a directory. Try it on */usr/share* — there is a lot of interesting stuff in there.
Although there are minor differences between Linux distributions, the layout for their filesystems are mercifully similar. So much so that you could say: once you know one, you know them all. And the best way to know the filesystem is to explore it. So go forth with tree, ls, and cd into uncharted territory.
You cannot damage your filesystem just by looking at it, so move from one directory to another and take a look around
Material adapted from https://www.linux.com/training-tutorials/linux-filesystem-explained/

CHAPTER 3

Basic Overview Of Linux Commands

echo command

NAME
echo - display a line of text
SYNOPSIS
echo [SHORT-OPTION]... [STRING]...
echo LONG-OPTION
DESCRIPTION
Echo the STRING(s) to standard output.
-n do not output the trailing newline

-e enable interpretation of backslash escapes
-E disable interpretation of backslash escapes (default)
--help display this help and exit
--version output version information and exit
If -e is in effect, the following sequences are recognized:
\\ backslash
\a alert (BEL)
\b backspace
\c produce no further output
\e escape
\f form feed
\n new line
\r carriage return
\t horizontal tab
\v vertical tab
\0NNN byte with octal value NNN (1 to 3 digits)
\xHH byte with hexadecimal value HH (1 to 2 digits)

read command

NAME
read - read from a file descriptor
SYNOPSIS
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);

DESCRIPTION
read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf. On files that support seeking, the read operation commences at the current file offset, and the file offset is incremented by the number of bytes read. If the current file offset is at or past the end of file, no bytes are read, and read() returns zero.
If count is zero, read() may detect the errors described below. In the absence of any errors, or if read() does not check for errors, a read() with a count of 0 returns zero and has no other effects. If count is greater than SSIZE_MAX, the result is unspecified. RETURN VALUE
On success, the number of bytes read is returned (zero indicates end of file), and the file position is advanced by this number. It is not an error if this number is smaller than the number of bytes requested; this may happen for example because fewer bytes are actually available right now (maybe because we were close to end-of-file, or because we are reading from a pipe, or from a terminal), or because read() was interrupted by a signal. See also NOTES. On error, -1 is returned, and errno is set appropriately. In this case, it is left unspecified whether the file position (if any) changes.
ERRORS
EAGAIN The file descriptor fd refers to a file other than a socket and has been marked nonblocking (O_NONBLOCK), and the read would block. See open(2) for further details on the O_NONBLOCK flag.
EAGAIN or EWOULDBLOCK
The file descriptor fd refers to a socket and has been marked

nonblockingO_NONBLOCK), and the read would block. POSIX.1-2001 allows either error to be returned for this case, and does not require these constants to have the same value, so a portable application should check for both possibilities.
EBADF fd is not a valid file descriptor or is not open for reading.
EFAULT buf is outside your accessible address space.
EINTR The call was interrupted by a signal before any data was read; see signal(7). EINVAL fd is attached to an object which is unsuitable for reading; or the file was opened with the O_DIRECT flag, and either the address specified in buf, the value specified in count, or the current file offset is not suitably aligned.
EINVAL fd was created via a call to timerfd_create(2) and the wrong size buffer was given to read(); see timerfd_create(2) for further information.
EIO I/O error. This will happen for example when the process is in a background process group, tries to read from its controlling terminal, and either it is ignoring or blocking SIGTTIN or its process group is orphaned. It may also occur when there is a low-level I/O error while reading from a disk or tape.
EISDIR fd refers to a directory.
Other errors may occur, depending on the object connected to fd. POSIX allows a read() that is interrupted after reading some data to return -1 (with errno set to EINTR) or to return the number of bytes already read.

more command

more command is used to display text in the terminal screen. It allows only backward movement.

The Syntax is

more [OPTIONS] filename

OPTIONS:

-c

Clear screen before displaying.

-e

Exit immediately after writing the last line of the last file in the argument list.

-n

Specify how many lines are printed in the screen for a given file.

+n

Starts up the file from the given number.

less command

less command is used to display text in the terminal screen. It just prints the text in the given file, you cannot edit or manipulate the text here. To display the file from the specified line, enter the line number followed by colon(:). It allows Forward and backward movement in the file.

The Syntax is

less [OPTIONS] filename

OPTIONS:

-c
Clear screen before displaying.
+n
Starts up the file from the given number.
:p
Examine the pervious file in the command line list.
:d
Remove the current file from the list of files.

man command

Type man followed by a command (for which you want help) and start reading. Press q to quit the manpage. Some man pages contain examples (near the end).

```
paul@laika:~$ man whois
Reformatting whois(1), please wait...
```

man $configfile

Most configuration files have their own manual.

```
paul@laika:~$ man syslog.conf
Reformatting syslog.conf(5), please wait...
```

man $daemon

This is also true for most daemons (background programs) on your system.. paul@laika:~$ man syslogd
Reformatting syslogd(8), please wait...

man -k (apropos)

man -k (or apropos) shows a list of man pages containing a string.

```
paul@laika:~$ man -k syslog
```

```
lm-syslog-setup (8) - configure laptop mode to switch syslog.conf ...
logger (1) - a shell command interface to the syslog(3) ...
syslog-facility (8) - Setup and remove LOCALx facility for sysklogd
syslog.conf (5) -syslogd(8) configuration file
syslogd (8) - Linux system logging utilities.
syslogd-listfiles (8) - list system logfiles
```

man sections

By now you will have noticed the numbers between the round brackets. man man will explain to you that these are section numbers. Executable programs and shell commands reside in section one.
1 Executable programs or shell commands
2 System calls (functions provided by the kernel)
3 Library calls (functions within program libraries)
4 Special files (usually found in /dev)
5 File formats and conventions eg /etc/passwd
6 Games
7 Miscellaneous (including macro packages and conventions), e.g.
man(7) 8 System administration commands (usually only for root)

9 Kernel routines [Non standard]

man $section $file

Therefor, when referring to the man page of the passwd command, you will see it written as passwd(1); when referring to the passwd file, you will see it written as passwd(5). The screenshot explains how to open the man page in the correct section.

```
[paul@RHEL52 ~]$ man passwd # opens the first manual found
[paul@RHEL52 ~]$ man 5 passwd # opens a page from section 5
```

man man

If you want to know more about man, then Read The Fantastic Manual (RTFM). Unfortunately, manual pages do not have the answer to everything...

```
paul@laika:~$ man woman
No manual entry for woman
```

man pages
25

mandb

Should you be convinced that a man page exists, but you can't access it, then try running mandb.

```
root@laika:~# mandb
0 man subdirectories contained newer manual pages.
0 manual pages were added.
0 stray cats were added.
0 old database entries were purged.
```

file ownership

user owner and group owner The users and groups of a system can be locally managed in /etc/passwd and /etc/ group, or they can be in a NIS, LDAP, or Samba domain. These users and groups can own files. Actually, every file has a user owner and a group owner, as can be seen in the following screenshot.

```
paul@RHELv4u4:~/test$ ls -l
total 24
-rw-rw-r-- 1 paul paul 17 Feb 7 11:53 file1
-rw-rw-r-- 1 paul paul 106 Feb 5 17:04 file2
-rw-rw-r-- 1 paul proj 984 Feb 5 15:38 data.odt
-rw-r--r-- 1 root root 0 Feb 7 16:07 stuff.txt
paul@RHELv4u4:~/test$
```

User paul owns three files, two of those are also owned by the group paul; data.odt is owned by the group proj. The root user owns the file stuff.txt, as does the group root.

Chown command

The user owner of a file can be changed with chown command.

```
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 root paul 0 2008-08-06 14:11 FileForPaul
root@laika:/home/paul# chown paul FileForPaul
root@laika:/home/paul# ls -l FileForPaul
```

```
-rw-r--r-- 1 paul paul 0 2008-08-06 14:11 FileForPaul
```

You can also use chown to change both the user owner and the group owner.

```
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 paul paul 0 2008-08-06 14:11 FileForPaul
root@laika:/home/paul# chown root:project42 FileForPaul
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 root project42 0 2008-08-06 14:11 FileForPaul
```

standard file permissions

list of special files When you use ls -l, for each file you can see ten characters before the user and group owner. The first character tells us the type of file. Regular files get a -, directories get a d, symbolic links are shown with an l, pipes get a p, character devices a c, block devices a b, and sockets an s.

Unix special files

-	:normal file
d	:directory
l	:symbolic link
p:	named pipe
b	:block device
c	:character device
s:	socket

Below a screenshot of a character device (the console) and a block device (the hard disk).

```
paul@debian6lt~$ ls -ld /dev/console /dev/sda
crw------- 1 root root 5, 1 Mar 15 12:45 /dev/console
brw-rw---- 1 root disk 8, 0 Mar 15 12:45 /dev/sda
```

And here you can see a directory, a regular file and a symbolic link.

```
paul@debian6lt~$ ls -ld /etc /etc/hosts /etc/motd
drwxr-xr-x 128 root root 12288 Mar 15 18:34 /etc
-rw-r--r-- 1 root root 372 Dec 10 17:36 /etc/hosts
lrwxrwxrwx 1 root root 13 Dec 5 10:36 /etc/motd -> /var/run/motd
```

standard file permissions

Permissions

rwx

The nine characters following the file type denote the permissions in three triplets. A permission can be r for read access, w for write access, and x for execute. You need the r permission to list (ls) the contents of a directory. You need the x permission to enter (cd) a directory. You need the w permission to create files in or remove files from a directory.

permission on a file on a directory

r (read) read file contents (cat) read directory contents (ls)
w (write) change file contents (vi) create files in (touch)
x (execute) execute the file enter the directory (cd)

three sets of rwx

We already know that the output of ls -l starts with ten characters for each file. This screenshot shows a regular file (because the first character is a -).

```
paul@RHELv4u4:~/test$ ls -l proc42.bash
-rwxr-xr-- 1 paul proj 984 Feb 6 12:01 proc42.bash
```

Below is a table describing the function of all ten characters.

file permissions position

position	characters	function
1	-	this is a regular file
2-4	rwx	permissions for the user owner
5-7	r-x	permissions for the group owner
8-10	r--	permissions for others

When you are the user owner of a file, then the user owner permissions apply to you. The rest of the permissions have no influence on your access to the file. When you belong to the group that is the group owner of a file, then the group owner permissions apply to you. The rest of the permissions have no influence on your access to the file. When you are not the user owner of a file and you do not belong to the group owner, then the others permissions apply to you. The rest of the permissions have no influence on your access to the file.

permission example:

Some example combinations on files and directories are seen in this screenshot. The name of the file explains the permissions.

```
paul@laika:~/perms$ ls -lh
total 12K
drwxr-xr-x 2 paul paul 4.0K 2007-02-07 22:26
AllEnter_UserCreateDelete -rwxrwxrwx 1 paul paul 0 2007-02-07 22:21
EveryoneFullControl.txt
-r--r----- 1 paul paul 0 2007-02-07 22:21 OnlyOwnersRead.txt
-rwxrwx--- 1 paul paul 0 2007-02-07 22:21 OwnersAll_RestNothing.txt
dr-xr-x--- 2 paul paul 4.0K 2007-02-07 22:25 UserAndGroupEnter
dr-x------ 2 paul paul 4.0K 2007-02-07 22:25 OnlyUserEnter
paul@laika:~/perms$
```

To summarise, the first rwx triplet represents the permissions for the user owner. The second triplet corresponds to the group owner; it specifies permissions for all members of that group. The third triplet defines permissions for all other users that are not the user owner and are not a member of the group owner.

Chmod command

Permissions can be changed with chmod. The first example gives the user owner execute

permissions.

```
paul@laika:~/perms$ ls -l permissions.txt
-rw-r--r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
paul@laika:~/perms$ chmod u+x permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwxr--r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

This example removes the group owners read permission.

```
paul@laika:~/perms$ chmod g-r permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx---r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

This example removes the others read permission.

```
paul@laika:~/perms$ chmod o-r permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx------ 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

This example gives all of them the write permission.

```
paul@laika:~/perms$ chmod a+w permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx-w--w- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

You don't even have to type the a.

```
paul@laika:~/perms$ chmod +x permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx-wx-wx 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

You can also set explicit permissions.

```
paul@laika:~/perms$ chmod u=rw permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rw--wx-wx 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

Feel free to make any kind of combination.

```
paul@laika:~/perms$ chmod u=rw,g=rw,o=r permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rw-rw-r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

Even fishy combinations are accepted by chmod.

```
paul@laika:~/perms$ chmod u=rwx,ug+rw,o=r permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwxrw-r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

setting octal permissions

Most Unix administrators will use the old school octal system to talk about and set permissions. Look at the triplet bitwise, equating r to 4, w to 2, and x to 1.

binary	octal	permission
000	0	---
001	1	--x
010	2	-w
011	3	-wx
100	4	r--
101	5	r-x
110	6	rw
111	7	rwx

This makes 777 equal to rwxrwxrwx and by the same logic, 654 mean rw-r-xr-- . The

chmod command will accept these numbers.

```
paul@laika:~/perms$ chmod 777 permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwxrwxrwx 1 paul paul 0 2007-02-07 22:34 permissions.txt
paul@laika:~/perms$ chmod 664 permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rw-rw-r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
paul@laika:~/perms$ chmod 750 permissions.txt
```

```
paul@laika:~/perms$ ls -l permissions.txt
-rwxr-x--- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

cd command

You can change your current directory with the cd command (Change Directory). paul@laika$ cd /etc

```
paul@laika$ pwd
/etc
paul@laika$ cd /bin
paul@laika$ pwd
/bin
paul@laika$ cd /home/paul/
paul@laika$ pwd
/home/paul
```

cd ~

You can pull off a trick with cd. Just typing cd without a target directory, will put you in your home directory. Typing cd ~ has the same effect.

```
paul@laika$ cd /etc
paul@laika$ pwd
/etc
paul@laika$ cd
paul@laika$ pwd
/home/paul
paul@laika$ cd ~
paul@laika$ pwd
/home/paul
```

cd ..

To go to the parent directory (the one just above your current directory in the directory tree), type cd .. .

```
paul@laika$ pwd
/usr/share/games
paul@laika$ cd ..
paul@laika$ pwd
/usr/share
```

To stay in the current directory, type cd . ;-) We will see useful use of the . character representing the current directory later.

working with directories

28

cd -

Another useful shortcut with cd is to just type cd - to go to the previous directory. paul@laika$ pwd

```
/home/paul
paul@laika$ cd /etc
paul@laika$ pwd
/etc
```

```
paul@laika$ cd -
/home/paul
paul@laika$ cd -
/etc
```

mkdir

Walking around the Unix file tree is fun, but it is even more fun to create your own directories with mkdir. You have to give at least one parameter to mkdir, the name of the new directory to be created. Think before you type a leading / .

```
paul@laika:~$ mkdir MyDir
paul@laika:~$ cd MyDir
paul@laika:~/MyDir$ ls -al
total 8
drwxr-xr-x 2 paul paul 4096 2007-01-10 21:13 .
drwxr-xr-x 39 paul paul 4096 2007-01-10 21:13 ..
paul@laika:~/MyDir$ mkdir stuff
paul@laika:~/MyDir$ mkdir otherstuff
paul@laika:~/MyDir$ ls -l
total 8
drwxr-xr-x 2 paul paul 4096 2007-01-10 21:14 otherstuff
drwxr-xr-x 2 paul paul 4096 2007-01-10 21:14 stuff
paul@laika:~/MyDir$
```

mkdir -p

When given the option -p, then mkdir will create parent directories as needed. paul@laika:~$ mkdir -p MyDir2/MySubdir2/ThreeDeep

```
paul@laika:~$ ls MyDir2
MySubdir2
paul@laika:~$ ls MyDir2/MySubdir2
ThreeDeep
paul@laika:~$ ls MyDir2/MySubdir2/ThreeDeep/
```

pwd

The you are here sign can be displayed with the pwd command (Print Working Directory). Go ahead, try it: Open a command line interface (like gnome-terminal, konsole, xterm, or a tty) and type pwd. The tool displays your current directory. paul@laika:~$ pwd

```
/home/paul
```

ls

You can list the contents of a directory with ls.

```
paul@pasha:~$ ls
allfiles.txt dmesg.txt httpd.conf stuff summer.txt
```

```
paul@pasha:~$
```

ls -a

A frequently used option with ls is -a to show all files. Showing all files means including the hidden files. When a file name on a Unix file system starts with a dot, it is considered a hidden file and it doesn't show up in regular file listings.

```
paul@pasha:~$ ls
allfiles.txt dmesg.txt httpd.conf stuff summer.txt
paul@pasha:~$ ls -a
. allfiles.txt .bash_profile dmesg.txt .lesshst stuff
.. .bash_history .bashrc httpd.conf .ssh summer.txt
paul@pasha:~$
```

ls -l

Many times you will be using options with ls to display the contents of the directory in different formats or to display different parts of the directory. Typing just ls gives you a list of files in the directory. Typing ls -l (that is a letter L, not the number 1) gives you a long listing.

```
paul@pasha:~$ ls -l
total 23992
-rw-r--r-- 1 paul paul 24506857 2006-03-30 22:53 allfiles.txt
-rw-r--r-- 1 paul paul 14744 2006-09-27 11:45 dmesg.txt
-rw-r--r-- 1 paul paul 8189 2006-03-31 14:01 httpd.conf
drwxr-xr-x 2 paul paul 4096 2007-01-08 12:22 stuff
-rw-r--r-- 1 paul paul 0 2006-03-30 22:45 summer.txt
```

ls -lh

Another frequently used ls option is -h. It shows the numbers (file sizes) in a more human readable format. Also shown below is some variation in the way you can give the options to ls. We will explain the details of the output later in this book.

```
paul@pasha:~$ ls -l -h
total 24M
-rw-r--r-- 1 paul paul 24M 2006-03-30 22:53 allfiles.txt
-rw-r--r-- 1 paul paul 15K 2006-09-27 11:45 dmesg.txt
-rw-r--r-- 1 paul paul 8.0K 2006-03-31 14:01 httpd.conf
drwxr-xr-x 2 paul paul 4.0K 2007-01-08 12:22 stuff
-rw-r--r-- 1 paul paul 0 2006-03-30 22:45 summer.txt
paul@pasha:~$ ls -lh
total 24M
-rw-r--r-- 1 paul paul 24M 2006-03-30 22:53 allfiles.txt
-rw-r--r-- 1 paul paul 15K 2006-09-27 11:45 dmesg.txt
-rw-r--r-- 1 paul paul 8.0K 2006-03-31 14:01 httpd.conf
drwxr-xr-x 2 paul paul 4.0K 2007-01-08 12:22 stuff
-rw-r--r-- 1 paul paul 0 2006-03-30 22:45 summer.txt
paul@pasha:~$ ls -hl
total 24M
-rw-r--r-- 1 paul paul 24M 2006-03-30 22:53 allfiles.txt
-rw-r--r-- 1 paul paul 15K 2006-09-27 11:45 dmesg.txt
-rw-r--r-- 1 paul paul 8.0K 2006-03-31 14:01 httpd.conf
```

```
drwxr-xr-x 2 paul paul 4.0K 2007-01-08 12:22 stuff
```

```
-rw-r--r-- 1 paul paul 0 2006-03-30 22:45 summer.txt
paul@pasha:~$ ls -h -l
total 24M
-rw-r--r-- 1 paul paul 24M 2006-03-30 22:53 allfiles.txt
-rw-r--r-- 1 paul paul 15K 2006-09-27 11:45 dmesg.txt
-rw-r--r-- 1 paul paul 8.0K 2006-03-31 14:01 httpd.conf
drwxr-xr-x 2 paul paul 4.0K 2007-01-08 12:22 stuff
-rw-r--r-- 1 paul paul 0 2006-03-30 22:45 summer.txt
```

find

The find command can be very useful at the start of a pipe to search for files. Here are some examples. You might want to add 2>/dev/null to the command lines to avoid cluttering your screen with error messages.

Find all files in /etc and put the list in etcfiles.txt

```
find /etc > etcfiles.txt
```

Find all files of the entire system and put the list in allfiles.txt

```
find / > allfiles.txt
```

Find files that end in .conf in the current directory (and all subdirs).

```
find . -name "*.conf"
```

Find files of type file (not directory, pipe or etc.) that end in .conf.

```
find . -type f -name "*.conf"
```

Find files of type directory that end in .bak .

```
find /data -type d -name "*.bak"
```

Find files that are newer than file42.txt

```
find . -newer file42.txt
```

Find can also execute another command on every file found. This example will look for *.odf files and copy them to /backup/.

```
find /data -name "*.odf" -exec cp {} /backup/ \;
```

Find can also execute, after your confirmation, another command on every file found. This example will remove *.odf files if you approve of it for every file found.

```
find /data -name "*.odf" -ok rm {} \;
```

cat

The cat command is one of the most universal tools. All it does is copy standard input to standard output. In combination with the shell this can be very powerful and diverse. Some examples will give a glimpse into the possibilities. The first example is simple, you can use cat to display a file on the screen. If the file is longer than the screen, it will scroll to the end. paul@laika:~$ cat /etc/resolv.conf

```
nameserver 194.7.1.4
paul@laika:~$
```

concatenate

cat is short for concatenate. One of the basic uses of cat is to concatenate files into a bigger (or complete) file.

```
paul@laika:~$ echo one > part1
paul@laika:~$ echo two > part2
paul@laika:~$ echo three > part3
paul@laika:~$ cat part1 part2 part3
```

```
one
two
three
paul@laika:~$
```

create files

You can use cat to create flat text files. Type the cat > winter.txt command as shown in the screenshot below. Then type one or more lines, finishing each line with the enter key. After the last line, type and hold the Control (Ctrl) key and press d.

```
paul@laika:~/test$ cat > winter.txt
It is very cold today!
paul@laika:~/test$ cat winter.txt
It is very cold today!
paul@laika:~/test$
```

The Ctrl d key combination will send an EOF (End of File) to the running process ending the cat command.

custom end marker

You can choose an end marker for cat with << as is shown in this screenshot. This construction is called a here directive and will end the cat command.

```
paul@laika:~/test$ cat > hot.txt <<stop
> It is hot today!
> Yes it is summer.
> stop
paul@laika:~/test$ cat hot.txt
It is hot today!
Yes it is summer.
paul@laika:~/test$
```

copy files

In the third example you will see that cat can be used to copy files. We will explain in detail what happens here in the bash shell chapter.

```
paul@laika:~/test$ cat winter.txt
It is very cold today!
paul@laika:~/test$ cat winter.txt > cold.txt
paul@laika:~/test$ cat cold.txt
It is very cold today!
paul@laika:~/test$
```

mv

Use mv to rename a file or to move the file to another directory.

```
paul@laika:~/test$ touch file100
paul@laika:~/test$ ls
file100
paul@laika:~/test$ mv file100 ABC.txt
paul@laika:~/test$ ls
ABC.txt
paul@laika:~/test$
```

When you need to rename only one file then mv is the preferred command to use.

cp

To copy a file, use cp with a source and a target argument. If the target is a directory, then the source files are copied to that target directory.

```
paul@laika:~/test$ touch FileA
paul@laika:~/test$ ls
FileA
paul@laika:~/test$ cp FileA FileB
paul@laika:~/test$ ls
FileA FileB
paul@laika:~/test$ mkdir MyDir
paul@laika:~/test$ ls
FileA FileB MyDir
paul@laika:~/test$ cp FileA MyDir/
paul@laika:~/test$ ls MyDir/
FileA
```

cp -r

To copy complete directories, use cp -r (the -r option forces recursive copying of all files in all subdirectories).

```
paul@laika:~/test$ ls
FileA FileB MyDir
paul@laika:~/test$ ls MyDir/
FileA
paul@laika:~/test$ cp -r MyDir MyDirB
paul@laika:~/test$ ls
FileA FileB MyDir MyDirB
paul@laika:~/test$ ls MyDirB
FileA
```

cp multiple files to directory

You can also use cp to copy multiple files into a directory. In this case, the last argument (a.k.a. the target) must be a directory.

```
cp file1 file2 dir1/file3 dir1/file55 dir2
```

cp -i

To prevent cp from overwriting existing files, use the -i (for interactive) option. paul@laika:~/test$ cp fire water

```
paul@laika:~/test$ cp -i fire water
cp: overwrite `water'? no
paul@laika:~/test$
```

cp -p

To preserve permissions and time stamps from source files, use cp -p. paul@laika:~/perms$ cp file* cp

```
paul@laika:~/perms$ cp -p file* cpp
```

```
paul@laika:~/perms$ ll *
-rwx------ 1 paul paul 0 2008-08-25 13:26 file33
-rwxr-x--- 1 paul paul 0 2008-08-25 13:26 file42
cp:
```

```
total 0
-rwx------ 1 paul paul 0 2008-08-25 13:34 file33
-rwxr-x--- 1 paul paul 0 2008-08-25 13:34 file42
cpp:
total 0
-rwx------ 1 paul paul 0 2008-08-25 13:26 file33
-rwxr-x--- 1 paul paul 0 2008-08-25 13:26 file42
```

rm

When you no longer need a file, use rm to remove it. Unlike some graphical user interfaces, the command line in general does not have a waste bin or trash can to recover files. When you use rm to remove a file, the file is gone. Therefore, be careful when removing files!

```
paul@laika:~/test$ ls
BigBattle SinkoDeMayo
paul@laika:~/test$ rm BigBattle
paul@laika:~/test$ ls
SinkoDeMayo
```

rm -i

To prevent yourself from accidentally removing a file, you can type rm -i.

```
paul@laika:~/Linux$ touch brel.txt
paul@laika:~/Linux$ rm -i brel.txt
rm: remove regular empty file `brel.txt'? y
paul@laika:~/Linux$
```

rm -rf

By default, rm -r will not remove non-empty directories. However rm accepts several options that will allow you to remove any directory. The rm -rf statement is famous because it will erase anything (providing that you have the permissions to do so). When you are logged on as root, be very careful with rm -rf (the f means force and the r means recursive) since being root implies that permissions don't apply to you. You can literally erase your entire file system by accident. paul@laika:~$ ls test

```
SinkoDeMayo
paul@laika:~$ rm test
rm: cannot remove `test': Is a directory
paul@laika:~$ rm -rf test
paul@laika:~$ ls test
ls: test: No such file or directory
```

wc

Counting words, lines and characters is easy with wc.

```
[paul@RHEL4b pipes]$ wc tennis.txt
```

```
5 15 100 tennis.txt
[paul@RHEL4b pipes]$ wc -l tennis.txt
5 tennis.txt
[paul@RHEL4b pipes]$ wc -w tennis.txt
15 tennis.txt
[paul@RHEL4b pipes]$ wc -c tennis.txt
```

```
100 tennis.txt
[paul@RHEL4b pipes]$
```

cut

The cut filter can select columns from files, depending on a delimiter or a count of bytes. The screenshot below uses cut to filter for the username and userid in the /etc/ passwd file. It uses the colon as a delimiter, and selects fields 1 and 3.

```
[[paul@RHEL4b pipes]$ cut -d: -f1,3 /etc/passwd | tail -4
Figo:510
Pfaff:511
Harry:516
Hermione:517
[paul@RHEL4b pipes]$
```

When using a space as the delimiter for cut, you have to quote the space. [paul@RHEL4b pipes]$ cut -d" " -f1 tennis.txt

```
Amelie
Kim
Justine
Serena
Venus
[paul@RHEL4b pipes]$
```

This example uses cut to display the second to the seventh character of /etc/passwd. [paul@RHEL4b pipes]$ cut -c2-7 /etc/passwd | tail -4

```
igo:x:
faff:x
arry:x
ermion
[paul@RHEL4b pipes]$
```

Paste

paste [OPTION]... [FILE]...

Write lines consisting of the sequentially corresponding lines from each FILE, separated by TABs, to standard output.

With no FILE, or when FILE is -, read standard input.

Mandatory arguments to long options are mandatory for short options too. -d, --delimiters=LIST reuse characters from LIST instead of TABs

-s, --serial paste one file at a time instead of in parallel

-z, --zero-terminated line delimiter is NUL, not newline

--help display this help and exit

--version output version information and exit

head

You can use head to display the first ten lines of a file.

```
paul@laika:~$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
```

```
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
paul@laika:~$
```

The head command can also display the first n lines of a file.

```
paul@laika:~$ head -4 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
```

Head can also display the first n bytes.

```
paul@laika:~$ head -c4 /etc/passwd
rootpaul@laika:~$
```

tail

Similar to head, the tail command will display the last ten lines of a file.

```
paul@laika:~$ tail /etc/services
vboxd 20012/udp
binkp 24554/tcp # binkp fidonet protocol
asp 27374/tcp # Address Search Protocol
asp 27374/udp
csync2 30865/tcp # cluster synchronization tool
dircproxy 57000/tcp # Detachable IRC Proxy
tfido 60177/tcp # fidonet EMSI over telnet
fido 60179/tcp # fidonet EMSI over TCP
# Local services
paul@laika:~$
```

You can give tail the number of lines you want to see.

```
$ tail -3 count.txt
six
seven
eight
```

The tail command has other useful options, some of which we will use during this course.

grep

The grep filter is famous among Unix users. The most common use of grep is to filter lines of text containing (or not containing) a certain string.

```
[paul@RHEL4b pipes]$ cat tennis.txt
Amelie Mauresmo, Fra
Kim Clijsters, BEL
Justine Henin, Bel
Serena Williams, usa
Venus Williams, USA
[paul@RHEL4b pipes]$ cat tennis.txt | grep Williams
Serena Williams, usa
```

```
Venus Williams, USA
```

You can write this without the cat.

```
[paul@RHEL4b pipes]$ grep Williams tennis.txt
Serena Williams, usa
Venus Williams, USA
```

One of the most useful options of grep is grep -i which filters in a case insensitive way. [paul@RHEL4b pipes]$ grep Bel tennis.txt

```
Justine Henin, Bel
[paul@RHEL4b pipes]$ grep -i Bel tennis.txt
Kim Clijsters, BEL
Justine Henin, Bel
[paul@RHEL4b pipes]$
```

Another very useful option is grep -v which outputs lines not matching the string. [paul@RHEL4b pipes]$ grep -v Fra tennis.txt

```
Kim Clijsters, BEL
Justine Henin, Bel
Serena Williams, usa
Venus Williams, USA
[paul@RHEL4b pipes]$
```

And of course, both options can be combined to filter all lines not containing a case insensitive string.

```
[paul@RHEL4b pipes]$ grep -vi usa tennis.txt
Amelie Mauresmo, Fra
Kim Clijsters, BEL
Justine Henin, Bel
[paul@RHEL4b pipes]$
```

With grep -A1 one line after the result is also displayed.

```
paul@debian5:~/pipes$ grep -A1 Henin tennis.txt
Justine Henin, Bel
Serena Williams, usa
```

With grep -B1 one line before the result is also displayed.

```
paul@debian5:~/pipes$ grep -B1 Henin tennis.txt
Kim Clijsters, BEL
Justine Henin, Bel
```

With grep -C1 (context) one line before and one after are also displayed. All three options

(A,B, and C) can display any number of lines (using e.g. A2, B4 or C20).

```
paul@debian5:~/pipes$ grep -C1 Henin tennis.txt
Kim Clijsters, BEL
Justine Henin, Bel
Serena Williams, usa
```

expr

```
expr EXPRESSION
or: expr OPTION
--help display this help and exit
--version output version information and exit
Print the value of EXPRESSION to standard output. A blank line below
separates increasing precedence groups. EXPRESSION may be:
```

```
ARG1 | ARG2 ARG1 if it is neither null nor 0, otherwise ARG2
ARG1 & ARG2 ARG1 if neither argument is null or 0, otherwise 0
ARG1 < ARG2 ARG1 is less than ARG2
ARG1 <= ARG2 ARG1 is less than or equal to ARG2
ARG1 = ARG2 ARG1 is equal to ARG2
ARG1 != ARG2 ARG1 is unequal to ARG2
ARG1 >= ARG2 ARG1 is greater than or equal to ARG2
ARG1 > ARG2 ARG1 is greater than ARG2
ARG1 + ARG2 arithmetic sum of ARG1 and ARG2
ARG1 - ARG2 arithmetic difference of ARG1 and ARG2
ARG1 * ARG2 arithmetic product of ARG1 and ARG2
ARG1 / ARG2 arithmetic quotient of ARG1 divided by ARG2
ARG1 % ARG2 arithmetic remainder of ARG1 divided by ARG2
STRING : REGEXP anchored pattern match of REGEXP in STRING
match STRING REGEXP same as STRING : REGEXP
substr STRING POS LENGTH substring of STRING, POS counted from 1
index STRING CHARS index in STRING where any CHARS is found, or 0
length STRING length of STRING
+ TOKEN interpret TOKEN as a string, even if it is a
keyword like 'match' or an operator like '/'
( EXPRESSION ) value of EXPRESSION
Beware that many operators need to be escaped or quoted for shells. Comparisons are
arithmetic if both ARGs are numbers, else lexicographical. Pattern matches return the string
matched between \( and \) or null; if \( and \) are not used, they return the number of
characters matched or 0. Exit status is 0 if EXPRESSION is neither null nor 0, 1 if EXPRESSION is
null or 0, 2 if EXPRESSION is syntactically invalid, and 3 if an error occurred.
```

redirection

One of the powers of the Unix command line is the use of redirection and pipes. This chapter first explains redirection of input, output and error streams. It then introduces pipes that consist of several commands.

stdin, stdout, and stderr

The shell (and almost every other Linux command) takes input from stdin (stream 0) and sends output to stdout (stream 1) and error messages to stderr (stream 2) . The keyboard often serves as stdin, stdout and stderr both go to the display. The shell allows you to redirect these streams.

output redirection

> stdout

stdout can be redirected with a greater than sign. While scanning the line, the shell will see the > sign and will clear the file.

```
[paul@RHELv4u3 ~]$ echo It is cold today!
It is cold today!
[paul@RHELv4u3 ~]$ echo It is cold today! > winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
It is cold today!
[paul@RHELv4u3 ~]$
```

Note that the > notation is in fact the abbreviation of 1> (stdout being referred to as stream 1. output file is erased To repeat: While scanning the line, the shell will see the > sign and will

clear the file! This means that even when the command fails, the file will be cleared!

```
[paul@RHELv4u3 ~]$ cat winter.txt
It is cold today!
[paul@RHELv4u3 ~]$ zcho It is cold today! > winter.txt
-bash: zcho: command not found
[paul@RHELv4u3 ~]$ cat winter.txt
[paul@RHELv4u3 ~]$
```

noclobber

Erasing a file while using > can be prevented by setting the noclobber option.

```
[paul@RHELv4u3 ~]$ cat winter.txt
It is cold today!
[paul@RHELv4u3 ~]$ set -o noclobber
[paul@RHELv4u3 ~]$ echo It is cold today! > winter.txt
-bash: winter.txt: cannot overwrite existing file
[paul@RHELv4u3 ~]$ set +o noclobber
[paul@RHELv4u3 ~]$
```

overruling noclobber

The noclobber can be overruled with >|.

```
[paul@RHELv4u3 ~]$ set -o noclobber
[paul@RHELv4u3 ~]$ echo It is cold today! > winter.txt
-bash: winter.txt: cannot overwrite existing file
[paul@RHELv4u3 ~]$ echo It is very cold today! >| winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
It is very cold today!
[paul@RHELv4u3 ~]$
```

>> append

Use >> to append output to a file.

```
[paul@RHELv4u3 ~]$ echo It is cold today! > winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
```

```
It is cold today!
[paul@RHELv4u3 ~]$ echo Where is the summer ? >> winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
It is cold today!
Where is the summer ?
[paul@RHELv4u3 ~]$
```

error redirection

2> stderr

Redirecting stderr is done with 2>. This can be very useful to prevent error messages from cluttering your screen. The screenshot below shows redirection of stdout to a file, and stderr to /dev/null. Writing 1> is the same as >.

```
[paul@RHELv4u3 ~]$ find / > allfiles.txt 2> /dev/null
[paul@RHELv4u3 ~]$
```

2>&1

To redirect both stdout and stderr to the same file, use 2>&1.

```
[paul@RHELv4u3 ~]$ find / > allfiles_and_errors.txt 2>&1
[paul@RHELv4u3 ~]$
```

Note that the order of redirections is significant. For example, the command

```
ls > dirlist 2>&1
```

directs both standard output (file descriptor 1) and standard error (file descriptor 2) to the file dirlist, while the command

```
ls 2>&1 > dirlist
```

directs only the standard output to file dirlist, because the standard error made a copy of the standard output before the standard output was redirected to dirlist.

input redirection

< stdin

Redirecting stdin is done with < (short for 0<).

```
[paul@RHEL4b ~]$ cat < text.txt
one
two
[paul@RHEL4b ~]$ tr 'onetw' 'ONEZZ' < text.txt
ONE
ZZO
[paul@RHEL4b ~]$
```

<< here document

The here document (sometimes called here-is-document) is a way to append input until a certain sequence (usually EOF) is encountered. The EOF marker can be typed literally or can be called with Ctrl-D.

```
[paul@RHEL4b ~]$ cat <<EOF > text.txt
> one
> two
> EOF
[paul@RHEL4b ~]$ cat text.txt
one
```

```
two
[paul@RHEL4b ~]$ cat <<brol > text.txt
> brel
> brol
[paul@RHEL4b ~]$ cat text.txt
brel
[paul@RHEL4b ~]$
```

<<< here string

The here string can be used to directly pass strings to a command. The result is the same as using echo string | command (but you have one less process running).

```
paul@ubu1110~$ base64 <<< linux-training.be
bGludXgtdHJhaW5pbmcuYmUK
paul@ubu1110~$ base64 -d <<< bGludXgtdHJhaW5pbmcuYmUK
linux-training.be
```

See rfc 3548 for more information about base64.

confusing redirection

The shell will scan the whole line before applying redirection. The following command line is very readable and is correct.

```
cat winter.txt > snow.txt 2> errors.txt
```

But this one is also correct, but less readable.

```
2> errors.txt cat winter.txt > snow.txt
```

Even this will be understood perfectly by the shell.

```
< winter.txt > snow.txt 2> errors.txt cat
```

pipes

One of the most powerful advantages of Linux is the use of pipes.
A pipe takes stdout from the previous command and sends it as stdin to the next command.
All commands in a pipe run simultaneously.

| vertical bar

Consider the following example.

```
paul@debian5:~/test$ ls /etc > etcfiles.txt
paul@debian5:~/test$ tail -4 etcfiles.txt
X11
xdg
xml
xpdf
paul@debian5:~/test$
```

This can be written in one command line using a pipe.

```
paul@debian5:~/test$ ls /etc | tail -4
X11
xdg
xml
xpdf
paul@debian5:~/test$
```

The pipe is represented by a vertical bar | between two commands.

multiple pipes

One command line can use multiple pipes. All commands in the pipe can run at the same time. paul@deb503:~/test$ ls /etc | tail -4 | tac

```
xpdf
xml
xdg
X11
```

users

user management

User management on any Unix can be done in three complimentary ways. You can use the graphical tools provided by your distribution. These tools have a look and feel that depends on the distribution. If you are a novice Linux user on your home system, then use the graphical tool that is provided by your distribution. This will make sure that you do not run into problems. Another option is to use command line tools like useradd, usermod, gpasswd, passwd and others. Server administrators are likely to use these tools, since they are familiar and very similar across many different distributions. This chapter will focus on these command line tools. A third and rather extremist way is to edit the local configuration files directly using vi (or vipw/vigr). Do not attempt this as a novice on production systems!

/etc/passwd

The local user database on Linux (and on most Unixes) is /etc/passwd.

```
[root@RHEL5 ~]# tail /etc/passwd
inge:x:518:524:art dealer:/home/inge:/bin/ksh
```

```
ann:x:519:525:flute player:/home/ann:/bin/bash
frederik:x:520:526:rubius poet:/home/frederik:/bin/bash
steven:x:521:527:roman emperor:/home/steven:/bin/bash
pascale:x:522:528:artist:/home/pascale:/bin/ksh
geert:x:524:530:kernel developer:/home/geert:/bin/bash
wim:x:525:531:master damuti:/home/wim:/bin/bash
sandra:x:526:532:radish stresser:/home/sandra:/bin/bash
annelies:x:527:533:sword fighter:/home/annelies:/bin/bash
laura:x:528:534:art dealer:/home/laura:/bin/ksh
```

As you can see, this file contains seven columns separated by a colon. The columns contain the username, an x, the user id, the primary group id, a description, the name of the home directory, and the login shell.

root

The root user also called the superuser is the most powerful account on your Linux system. This user can do almost anything, including the creation of other users. The root user always has userid 0 (regardless of the name of the account).

```
[root@RHEL5 ~]# head -1 /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

useradd

You can add users with the useradd command. The example below shows how to add a user named yanina (last parameter) and at the same time forcing the creation of the home directory (-m), setting the name of the home directory (-d), and setting a description (-c).

```
[root@RHEL5 ~]# useradd -m -d /home/yanina -c "yanina wickmayer" yanina [root@RHEL5
~]# tail -1 /etc/passwd
yanina:x:529:529:yanina wickmayer:/home/yanina:/bin/bash
```

The user named yanina received userid 529 and primary group id 529.

/etc/default/useradd

Both Red Hat Enterprise Linux and Debian/Ubuntu have a file called /etc/default/ useradd that contains some default user options. Besides using cat to display this file, you can also use useradd -D.

```
[root@RHEL4 ~]# useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/bash
SKEL=/etc/skel
```

userdel

You can delete the user yanina with userdel. The -r option of userdel will also remove the home directory.

```
[root@RHEL5 ~]# userdel -r yanina
```

usermod

You can modify the properties of a user with the usermod command. This example uses usermod to change the description of the user harry.

```
[root@RHEL4 ~]# tail -1 /etc/passwd
harry:x:516:520:harry potter:/home/harry:/bin/bash
```

```
[root@RHEL4 ~]# usermod -c 'wizard' harry
[root@RHEL4 ~]# tail -1 /etc/passwd
harry:x:516:520:wizard:/home/harry:/bin/bash
```

passwords

passwd

Passwords of users can be set with the passwd command. Users will have to provide their old password before twice entering the new one.

```
[harry@RHEL4 ~]$ passwd
Changing password for user harry.
Changing password for harry
(current) UNIX password:
New UNIX password:
BAD PASSWORD: it's WAY too short
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
[harry@RHEL4 ~]$
```

As you can see, the passwd tool will do some basic verification to prevent users from using too simple passwords. The root user does not have to follow these rules (there will be a warning though). The root user also does not have to provide the old password before entering the new password twice.

/etc/shadow

User passwords are encrypted and kept in /etc/shadow. The /etc/shadow file is read only and can only be read by root. We will see in the file permissions section how it is possible for users to change their password. For now, you will have to know that users can change their password with the /usr/bin/passwd command.

```
[root@RHEL5 ~]# tail /etc/shadow
inge:$1$yWMSimOV$YsYvcVKqByFVYLKnU3ncd0:14054:0:99999:7:::
ann:!!:14054:0:99999:7:::
frederik:!!:14054:0:99999:7:::
steven:!!:14054:0:99999:7:::
pascale:!!:14054:0:99999:7:::
geert:!!:14054:0:99999:7:::
wim:!!:14054:0:99999:7:::
sandra:!!:14054:0:99999:7:::
annelies:!!:14054:0:99999:7:::
laura:$1$Tvby1Kpa$lL.WzgobujUS3LClIRmdv1:14054:0:99999:7:::
```

The /etc/shadow file contains nine colon separated columns. The nine fields contain (from left to right) the user name, the encrypted password (note that only inge and laura have an encrypted password), the day the password was last changed (day 1 is January 1, 1970), number of days the password must be left unchanged, password expiry day, warning number of days before password expiry, number of days after expiry before disabling the account, and the day the account was disabled (again, since 1970). The last field has no meaning yet.

password encryption encryption with passwd

Passwords are stored in an encrypted format. This encryption is done by the crypt function. The easiest (and recommended) way to add a user with a password to the system is to add the user

with the useradd -m user command, and then set the user's password with passwd.

```
[root@RHEL4 ~]# useradd -m xavier
[root@RHEL4 ~]# passwd xavier
Changing password for user xavier.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
[root@RHEL4 ~]#
```

encryption with openssl

Another way to create users with a password is to use the -p option of useradd, but that option requires an encrypted password. You can generate this encrypted password with the openssl passwd command.

```
[root@RHEL4 ~]# openssl passwd stargate
ZZNX16QZVgUQg
[root@RHEL4 ~]# useradd -m -p ZZNX16QZVgUQg mohamed
```

encryption with crypt

A third option is to create your own C program using the crypt function, and compile this into a command.

```
[paul@laika ~]$ cat MyCrypt.c
#include <stdio.h>
#define __USE_XOPEN
#include <unistd.h>
int main(int argc, char** argv)
{
if(argc==3)
{
printf("%s\n", crypt(argv[1],argv[2]));
}
else
{
printf("Usage: MyCrypt $password $salt\n" );
}
return 0;
}
```

This little program can be compiled with gcc like this.

```
[paul@laika ~]$ gcc MyCrypt.c -o MyCrypt -lcrypt
```

To use it, we need to give two parameters to MyCript. The first is the unencrypted password, the second is the salt. The salt is used to perturb the encryption algorithm in one of 4096 different ways. This variation prevents two users with the same password from having the same entry in /etc/shadow.

```
paul@laika:~$ ./MyCrypt stargate 12
12L4FoTS3/k9U
paul@laika:~$ ./MyCrypt stargate 01
01Y.yPnlQ6R.Y
paul@laika:~$ ./MyCrypt stargate 33
330asFUbzgVeg
paul@laika:~$ ./MyCrypt stargate 42
```

```
42XFxoT4R75gk
```

Did you notice that the first two characters of the password are the salt? The standard output of the crypt function is using the DES algorithm which is old and can be cracked in minutes. A better method is to use md5 passwords which can be recognized by a salt starting with $1$.

```
paul@laika:~$ ./MyCrypt stargate '$1$12'
$1$12$xUIQ4116Us.Q5Osc2Khbm1
paul@laika:~$ ./MyCrypt stargate '$1$01'
$1$01$yNs8brjp4b4TEw.v9/IlJ/
paul@laika:~$ ./MyCrypt stargate '$1$33'
$1$33$tLh/Ldy2wskdKAJR.Ph4M0
paul@laika:~$ ./MyCrypt stargate '$1$42'
$1$42$Hb3nvP0KwHSQ7fQmIlY7R.
```

The md5 salt can be up to eight characters long. The salt is displayed in /etc/shadow between the second and third $, so never use the password as the salt!

```
paul@laika:~$ ./MyCrypt stargate '$1$stargate'
$1$stargate$qqxoLqiSVNvGr5ybMxEVM1
```

password defaults

/etc/login.defs

The /etc/login.defs file contains some default settings for user passwords like password aging and length settings. (You will also find the numerical limits of user ids and group ids and whether or not a home directory should be created by default).

```
[root@RHEL4 ~]# grep -i pass /etc/login.defs
# Password aging controls:
# PASS_MAX_DAYS Maximum number of days a password may be used.
# PASS_MIN_DAYS Minimum number of days allowed between password
changes. # PASS_MIN_LEN Minimum acceptable password length.
# PASS_WARN_AGE Number of days warning given before a password
expires. PASS_MAX_DAYS 99999
PASS_MIN_DAYS 0
PASS_MIN_LEN 5
PASS_WARN_AGE 7
```

chage

The chage command can be used to set an expiration date for a user account (-E), set a minimum (-m) and maximum (-M) password age, a password expiration date, and set the number of warning days before the password expiration date. Much of this functionality is also available from the passwd command. The -l option of chage will list these settings for a user.

```
[root@RHEL4 ~]# chage -l harry
Minimum: 0
Maximum: 99999
Warning: 7
Inactive: -1
Last Change: Jul 23, 2007
Password Expires: Never
Password Inactive: Never
Account Expires: Never
[root@RHEL4 ~]#
```

disabling a password

Passwords in /etc/shadow cannot begin with an exclamation mark. When the second field in /etc/passwd starts with an exclamation mark, then the password can not be used. Using this feature is often called locking, disabling, or suspending a user account.Besides vi (or vipw) you can also accomplish this with usermod.The first line in the next screenshot will disable the password of user harry, making it impossible for harry to authenticate using this password.

```
[root@RHEL4 ~]# usermod -L harry
[root@RHEL4 ~]# tail -1 /etc/shadow
harry:!$1$143TO9IZ$RLm/FpQkpDrV4/Tkhku5e1:13717:0:99999:7:::
```

The root user (and users with sudo rights on su) still will be able to su to harry (because the password is not needed here). Also note that harry will still be able to login if he has set up passwordless ssh!

```
[root@RHEL4 ~]# su - harry
[harry@RHEL4 ~]$
```

You can unlock the account again with usermod -U. Watch out for tiny differences in the command line options of passwd, usermod, and useradd on different distributions! Verify the local files when using features like "disabling, suspending, or locking" users and passwords! editing local files If you still want to manually edit the /etc/passwd or /etc/shadow, after knowing these commands for password management, then use vipw instead of vi(m) directly. The vipw tool will do proper locking of the file.

```
[root@RHEL5 ~]# vipw /etc/passwd
vipw: the password file is busy (/etc/ptmp present)
```

tar command

The tar command stands for tape achieve, which is the most commonly used tape drive backup command used by the Linux/Unix system. It allows for you to quickly access a collection of files and placed them into a highly compressed archive file commonly called tarball, or tar, gzip, and bzip in Linux. The algorithm used for compressing of .tar.gz and .tar.bz2 are gzip or bzip
algorithms respectively.

Extract a tar.gz archive

The following command shell helps to extract tar files out a tar.gz
archive. # tar -xvzf bigfile.tar.gz

x – Extract files
v – Verbose, print the file names as they are extracted one by one
z – The file is a “gzipped” file
f – Use the following tar archive for the operation

Extract files to a specific directory or path

We can extract the files into a specified directory by using the parameter
“-C”. # tar -xvzf bigfile.tar.gz -C /folder/subfolder/
The tar command doesn’t create any files or folders.

Extract a single file

To extract a single file out of an archive just add the file name after the command like
this # tar -xz -f Music.tar.gz “./new/one.mp3”
To extract more than one file out of an archive
#tar -xv -f Music.tar.gz “./new/two.mp3” “./new/three.mp3”

Extract multiple files using wildcards

To extract a group of files from archive
# tar -xv -f Music.tar.gz –wildcards “*.mp3”

List and search contents of the tar archive

To list out the tar archive without extracting, we can use ‘-t’
# tar -tz -f abc.tar.gz
For better viewing we can use less command or grep the pipe output for searching a
file. # tar -tvz -f Music.tar.gz | grep two.mp3
The verbose option “v” will provide more details about each file.

For tar.bz2/bzip files we should use “j” verbose option.

Create a tar/tar.gz archive

Using tar command, we can create tar archive using a directory, by adding all files in an archive or an entire directory.

```
# tar -cvf Videos.tar ./Latest/
```

The above command does not provide compression, but allows you to merge multiple files into one.

For the compression we could use the “z” or “j” option for gzip or bzip respectively. # tar -cvzf Videos.tar ./Latest/

The extension names does not have much more importance. What really matter is “.tar.gz” which is commonly used with gzip compression and .”tar.bz2″ and “.tbz”.They are commonly used extensions for bzip compressed files.

Permission before adding files

The option “w” allows tar asking for permission for archiving each file. The files which are replied by are only archived and with no reply will be considered as “No”.

```
# tar -czw -f hugefie.tar.gz ./Videos/*
```

The result of the above executed command is, which can be viewed by listing: # tar -t -f hugefie.tar.gz

Add files to existing archives

By using the “r” option it adds files to existing archives, without the creation of a new one. # tar -rv -f Book.tar pageone.txt

The main thing to be aware of is this option works only with normal archives, and not with the compressed archives.

Add files to compressed archives (tar.gz/tar.bz2)

It was mentioned earlier that it is not possible to add files to a compressed archive. However, it can be done by performing a simple step. Use the gunzip command to uncompress the archive, add the file to the archive and finally compress it.

```
# gunzip Institution.tar.gz
# tar -rf Institution.tar ./College/Engineering/top.ppt
# gzip Institution.tar
```

SAMPLE QUESTIONS

1. Command to display the following message (Use " and Newline).
"God! Bless us..
We are starting Shell Scripting"
2. Read your name from the keyboard and display it.
3. Create the directory structure dir1/dir4 and dir1/dir2/dir3 with a single command and then change directory to dir3

4. create a le test le1 using nano. Display the File
1. starting with the rst 10 lines and
2. starting with the 10th line with provision for 1. Scrolling Up 2. Scrolling Up and Down
5. Get the manual page of 'ls' command. Search for the word "alphabetic". Find next occurrence and then nd previous occurrence.

6. Create 2 files test file2 and test file3 using nano.
7. Modify the permissions of test file2 using symbolic mode
Add read permission to others
revoke write from owner
set only execute to Group
add write to owner, revoke read rom others and set read only to group.
set read and write to all
8. Modify the permissions of test file3 using numeric mode
1. Set read and write to all
2. set read,write and execute to owner, read and execute to group
and read only to others
9. Set the permissions of test file2 the same as that of test file3
10.Set the permissions of the tree (thedirectory, its chidren, grand chil-dren, etc.) rooted at
dir1 (Qn. 3) directory to 664
11. Change the owner and group of the directory tree from dir2 to
student. 1. Display the current directory
12. Listing Files and folders
1. List the contents of dir1 (Qn. 3) and all its descendants
2. List the contents of dir3 (Qn. 3) in Alphabetical Order Time of modi cation,
newest rst Sorted on Size Reverse of all above Long listing of les Sorted on
Size with smallest rst and size in human readable form
13. 1. Execute ls and store the output to a le lsoutput
2. Execute ls -l and add the output to lsoutput, at the end.
14. Execute ls -l and feed the result to less command, to scroll through the
directory listing.
15. A. Create a le fie1 containing the word "Hello," using cat and output
redirection B. Create another file file2 containing the word "Greetings!!"
C. Display the sentence, Hello, yourname, Greetings!!, using cat, by
concatenating file1, Standard Input and file2
16. Copy the file file1 to new file.
1. If new file already exists, it should be replaced.
2. If new file already exists, it should not be replaced.
3. If new file already exists, it shouldnot be replaced, but only with the consent
of the user.
4. If new file already exists, it should be replaced only if its contents is older
than that of new file.
5. Even if new file is read only.
6. Create a link instead of copying.
7. Copy the entire directory tree from dir1 of to a new direc-tory dir5
17. Create a new dirctory, dir6 inside dir1
1. Move all files in dir5 into it.
2. Rename the le new file in Qn.4 to old file

38

3. Move the file file1 in Qn.4 to dir6 with the name fi le3.
4. Delete all files where name starts with a vowel character, upper or lower
case. 5. Delete all fi les where the name is at least 3 characters long.
6. Delete all hidden folders, and files.

18. Using cut filter
1. Display the file names from ls -l assuming filenames start at column 50. 2.
Display user Id and user name of all users from /etc/passwd. (elds 1 and 3)
19. Using tr
1. Piped with cat, display all that are entered in Uppercase.
2. Squeeze all space characters in ls -l and cut the size and name of all files. 20. Create 3
files containing name, age and marks of 5 students respectively and paste them into a
single csv file.
21. Using find
1. piped with wc, display the number of files in a directory that starts with the letter
a 2. Delete all .c files in the parent directory
3. Copy all files that starts with a to dir2
4. Display files in the current directory that were modified in the last 30
minutes. 22. Use head and tail piped with cat /etc/passwd to display the details
of 1. The first 12 users in the system.
2. The last 7 users in the system.
3. All but the first 3.
4. All but the last 5.
5. Only the 9 th .
23. Use grep to
1. Display all lines in a file that contains the string abc
2. Display all lines in a file that does not contain the string abc
3. List names of all .c files that contains a printf
4. List names of all .c files that does not contain a printf
5. Display the number of #include statements in each .c file.
6. Display the Line numbers of printf in a .c file.
7. List names of all files in the directory tree that contain a printf.
8. Display the context of every printf in a .c file. i:e:; n lines before and after
every printf.
9. ls -l starts with d for directories. Use ls -l piped with grep & cut to
display thenames of all directories in the current directory.
24. Using expr
(a) Read two integers X and Y . Display the sum, diference, product, quotient and remainder
of these variables.
(b) Read a string, S, a position, p and a length l. Display the substring of length l starting
at position p from the string S.
25. Using bc
Use two variables x and y initialised with 10.3 and 5.6 respectively. Display the sum, di
erence, product and quotient of these variables.
26. Add a normal user, user1. Create (if it does not exist) and set /user1 as the home
directory. Also set /bin/bash as the login shell.
(For executing commands requiring root previleges, use kvm. For more details, refer
http://mca.rit.ac.in/Publish/2016-19/S4/RLMCA232 /virtualmachine/Readme.txt) 27. Modify
the user account of user1, to expire it after a speci c date and permanently disable it

after 5 more days.
28. Change the ownership of dir1 and all its contents to

user1 29. Delete the user account user1
By retaining the home folder By deleting the home folder

Chapter 4
Shell Scripting

What is a Shell?

An Operating is made of many components, but its two prime components are
- ● Kernel
● Shell

A Kernel is at the nucleus of a computer. It makes the communication between the hardware and software possible. While the Kernel is the innermost part of an operating system, a shell is the outermost one. A shell in a Linux operating system takes input from you in the form of commands, processes it, and then gives an output. It is the interface through which a user works on the programs, commands, and scripts. A shell is accessed by a terminal which runs it. When you run the terminal, the Shell issues **a command prompt (usually $),** where you can type your input, which is then executed when you hit the Enter key. The output or the result is thereafter displayed on the terminal. The Shell wraps around the delicate interior of an Operating system protecting it from accidental damage. Hence the name **Shell** . **Types of Shell**

There are two main shells in Linux:
1 . The **Bourne Shell** : The prompt for this shell is $ and its derivatives are listed below: ● POSIX shell also is known as sh
● Korn Shell also knew as sh
● **B** ourne **A** gain **SH** ell also knew as bash (most popular)
2. The C shell : The prompt for this shell is %, and its subcategories are: ● C shell also is known as csh
● Tops C shell also is known as tcsh
We will discuss bash shell based shell scripting in this tutorial.

What is Shell Scripting?

Shell scripting is writing a series of command for the shell to execute. It can combine lengthy and repetitive sequences of commands into a single and simple script, which can be stored and executed anytime. This reduces the effort required by the end user.
Let us understand the steps in creating a Shell Script
1. **Create a file using** a **vi** editor(or any other editor). Name script file with **extension .sh**
2. **Start** the script with **#! /bin/sh**
3. Write some code.
4. Save the script file as filename.sh
5. For **executing** the script type **bash filename.sh**

"#!" is an operator called shebang which directs the script to the interpreter location. So, if we use"#! /bin/sh" the script gets directed to the bourne-shell.
Let's create a small script -

```
#!/bin/sh
ls
```

Let's see the steps to create it -
Command 'ls' is executed when we execute the scrip sample.sh file.

Adding shell comments

Commenting is important in any program. In Shell programming, the syntax to add a comment is #comment
Let understand this with an example.

What are Shell Variables?

As discussed earlier, Variables store data in the form of characters and numbers. Similarly, Shell variables are used to store information and they can by the shell only. For example, the following creates a shell variable and then prints it:

```
variable ="Hello"
echo $variable
```

Below is a small script which will use a variable.

```
#!/bin/sh
echo "what is your name?"
read name
echo "How do you do, $name?"
read remark
echo "I am $remark too!"
```

Let's understand, the steps to create and execute the script

Environment variables

In Linux and Unix based systems environment variables are a set of dynamic named values, stored within the system that are used by applications launched in shells or subshells. In simple words, an environment variable is a variable with a name and an associated value. Environment variables allow you to customize how the system works and the behavior of the applications on the system. For example, the environment variable can store information

about the default text editor or browser, the path to executable files, or the system locale and keyboard layout settings.
In this guide, we will explain to read and set environment and shell variables.

Environment Variables and Shell Variables

Variables have the following format:

```
KEY=value
KEY="Some other value"
KEY=value1:value2
```

Copy

● The names of the variables are case-sensitive. By convention, environment variables should have UPPER CASE names.
● When assigning multiple values to the variable they must be separated by the colon : character.
● There is no space around the equals = symbol.

Variables can be classified into two main categories, environment variables, and shell variables. **Environment variables** are variables that are available system-wide and are inherited by all spawned child processes and shells.
Shell variables are variables that apply only to the current shell instance. Each shell such as zsh and bash, has its own set of internal shell variables.
There are several commands available that allow you to list and set environment variables in Linux:

● env – The command allows you to run another program in a custom environment without modifying the current one. When used without an argument it will print a list of the current environment variables.
● printenv – The command prints all or the specified environment variables. ● set – The command sets or unsets shell variables. When used without an argument it will print a list of all variables including environment and shell variables, and shell functions.
● unset – The command deletes shell and environment variables.
● export – The command sets environment variables.

List Environment Variables

The most used command to displays the environment variables is printenv. If the name of the variable is passed as an argument to the command, only the value of that variable is displayed. If no argument is specified, printenv prints a list of all environment variables, one variable per
line. For example, to display the value of the HOME environment variable you would run: printenv HOME
The output will print the path of the currently logged in user:

```
/home/linuxize
```

You can also pass more than one arguments to the printenv command:

```
printenv LANG PWD
en_US
/home/linuxize
```

If you run the printenv or env command without any arguments it will show a list of all environment variables:

```
printenv
```

The output will look something like this:

```
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35;...
LESSCLOSE=/usr/bin/lesspipe %s %s
LANG=en_US
S_COLORS=auto
```

```
XDG_SESSION_ID=5
USER=linuxize
PWD=/home/linuxize
HOME=/home/linuxize
SSH_CLIENT=192.168.121.1 34422 22
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
SSH_TTY=/dev/pts/0
MAIL=/var/mail/linuxize
TERM=xterm-256color
SHELL=/bin/bash
SHLVL=1
LANGUAGE=en_US:
LOGNAME=linuxize
XDG_RUNTIME_DIR=/run/user/1000
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/sn
a p/bin
LESSOPEN=| /usr/bin/lesspipe %s
_=/usr/bin/printenv
```

Below are some of the most common environment variables:

● USER - The current logged in user.
● HOME - The home directory of the current user.
● EDITOR - The default file editor to be used. This is the editor that will be used when
you type edit in your terminal.
● SHELL - The path of the current user's shell, such as bash or zsh.
● LOGNAME - The name of the current user.
● PATH - A list of directories to be searched when executing commands. When you run
a command the system will search those directories in this order and use the first
found executable.
● LANG - The current locales settings.
● TERM - The current terminal emulation.
● MAIL - Location of where the current user's mail is stored.

The printenv and env commands print only the environment variables. If you want to get a list of all variables, including environment, shell and variables, and shell functions you can use the set command:

```
set
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:complete_fullquote:expand_aliases:extglob:extquote:force
_f ignore:histappend:interactive_comments:login_shell:progcomp:promptvars:sourcepath
BASH_ALIASES=()
```

BASH_ARGC=()
BASH_ARGV=()
The command will display a large list of all variables so you probably want to pipe the output to the less command.
set | less
You can also use the echo command to print a shell variable. For example, to print the value of the BASH_VERSION variable you would run:
echo $BASH_VERSION
4.4.19(1)-release

Setting Environment Variables

To better illustrate the difference between the Shell and Environment variables we'll start with setting Shell Variables and then move on to the Environment variables. To create a new shell variable with the name MY_VAR and value Linuxize simply type: MY_VAR='Linuxize' You can verify that the variable is set by using either echo $MY_VAR of filtering the output of the set command with grep set | grep MY_VAR: echo $MY_VAR Linuxize
Use the printenv command to check whether this variable is an environment variable or not: printenv MY_VAR The output will be empty which tell us that the variable is not an environment variable. You can also try to print the variable in a sub-shell and you will get an empty output. bash -c 'echo $MY_VAR' The export command is used to set Environment variables. To create an environment variable simply export the shell variable as an environment variable: export MY_VAR You can check this by running: printenv MY_VAR Linuxize If you try to print the variable in a sub-shell this time you will get the variable name printed on your terminal: bash -c 'echo $MY_VAR' Linuxize You can also set environment variables in a single line: export MY_NEW_VAR="My New Var"
Environment Variables created in this way are available only in the current session. If you open a new shell or if you log out all variables will be lost.

Persistent Environment Variables

To make Environment variables persistent you need to define those variables in the bash configuration files. In most Linux distributions when you start a new session, environment variables are read from the following files: /etc/environment - Use this file to set up system-wide environment variables. Variables in this file are set in the following format: OO=bar VAR_TEST="Test Var"
● /etc/profile - Variables set in this file are loaded whenever a bash login shell is entered. When declaring environment variables in this file you need to use the export command: export JAVA_HOME="/path/to/java/home" export PATH=$PATH:$JAVA_HOME/bin ● Per-user shell specific configuration files. For example, if you are using Bash, you can declare the variables in the ~/.bashrc:
export PATH="$HOME/bin:$PATH"
● To load the new environment variables into the current shell session use the source command: source ~/.bashrc

Conditional Statements | Shell Script

Conditional Statements: There are total 5 conditional statements which can be used in bash programming
1. if statement

2. if-else statement
3. if..elif..else..fi statement (Else If ladder)
4. if..then..else..if..then..fi..fi..(Nested if)
5. switch statement
Their description with syntax is as follows:

if statement

This block will process if specified condition is true.
Syntax:
```
if [ expression ]
then
```

```
statement
fi
```

if-else statement

If specified condition is not true in if part then else part will be execute.
Syntax
```
if [ expression ]
then
statement1
else
statement2
fi
```

if..elif..else..fi statement (Else If ladder)

To use multiple conditions in one if-else block, then elif keyword is used in shell. If expression1 is true then it executes statement 1 and 2, and this process continues. If none of the condition is true then it processes else part.
Syntax
```
if [ expression1 ]
then
statement1
statement2
.
elif [ expression2 ]
then
statement3
statement4
.
else
statement5
fi
```

if..then..else..if..then..fi..fi..(Nested if)

Nested if-else block can be used when, one condition is satisfies then it again checks another condition. In the syntax, if expression1 is false then it processes else part, and again expression2 will be check.

Syntax:

```
if [ expression1 ]
then
statement1
statement2 .
else
if [ expression2 ]
then
statement3
.
fi
fi
```

switch statement

case statement works as a switch statement if specified value match with the pattern then it will execute a block of that particular pattern

When a match is found all of the associated statements until the double semicolon (;;) is executed.

A case will be terminated when the last command is executed.

If there is no match, the exit status of the case is zero.

Syntax:

```
case in
Pattern 1) Statement 1;;
Pattern n) Statement n;;
esac
```

Example Programs

Example 1:

Implementing if statement

filter_none

brightness_4

```
#Initializing two variables
a=10
b=20
#Check whether they are equal
if [ $a == $b ]
then
echo "a is equal to b"
fi
#Check whether they are not
equal
if [ $a != $b ]
then
echo "a is not equal to b"
fi
```

```
Output
$bash -f main.sh
a is not equal to b
```

Example 2:

Implementing if.else statement

```
filter_none
brightness_4
#Initializing two variables
a=20
b=20
if [ $a == $b ]
then
#If they are equal then print this
echo "a is equal to b"
else
```

```
#else print this
echo "a is not equal to b"
fi
Output
$bash -f main.sh
a is equal to b
```

Example 3:

Implementing switch statement

```
filter_none
brightness_4
CARS="bmw"
#Pass the variable in string
case "$CARS" in
#case 1
"mercedes") echo "Headquarters - Affalterbach,
Germany" ;;
#case 2
"audi") echo "Headquarters - Ingolstadt, Germany" ;;
#case 3
"bmw") echo "Headquarters - Chennai, Tamil Nadu, India"
;;
esac
Output
$bash -f main.sh
Headquarters - Chennai, Tamil Nadu, India.
```

SAMPLE QUESTIONS

For each of the questions, give the Command, its syntax and a

description of the important options of the command.
1.
Using find
1. piped with wc, display the number of files in a directory that starts with the letter
a 2. Delete all .c files in the parent directory
3. Copy all files that starts with a to dir2
4. Display files in the current directory that were modified in the last 30
minutes. 2. Use head and tail piped with cat /etc/passwd to display the details
of 1. The first 12 users in the system.
2. The last 7 users in the system.
3. All but the first 3
4. All but the last 5.
5. Only the 9 th .
3. Use grep to
1. Display all lines in a file that contains the string abc
2. Display all lines in a file that does not contain the string abc
3. List names of all .c files that contains a printf
4. List names of all .c files that does not contain a printf

5. Display the number of #include statements in each .c file.
6. Display the Line numbers of printf in a .c file.
7. List names of all files in the directory tree that contain a printf. 8. Display the
context of every printf in a .c file. i:e:; n lines before and after every printf.
9. ls -l starts with d for directories. Use ls -l piped with grep & cut to
display thenames of all directories in the current directory.
4. Using expr
1. Read two integers X and Y . Display the sum, diference, product, quotient
and remainder of these variables.
2. Read a string, S, a position, p and a length l. Display the substring of
length l starting at position p from the string S.
5. Using bc
Use two variables x and y initialised with 10.3 and 5.6 respectively. Display the sum, di erence, product
and quotient of these variables.
6. Add a normal user, user1. Create (if it does not exist) and set /user1 as
the home directory. Also set /bin/bash as the login shell.
(For executing commands requiring root previleges, use kvm. For more details,
refer http://mca.rit.ac.in/Publish/2016-19/S4/RLMCA232
/virtualmachine/Readme.txt) 7. Modify the user account of user1, to expire it
after a speci c date and permanently disable it after 5 more days.
8. Change the ownership of dir1 and all its contents to user1
9. Delete the user account user1
By retaining the home folder
By deleting the home folder

Chapter 5
Remote Access

Many times we need to work with remote Linux systems. We login to the remote host, perform work and exit that session. Can we perform all these actions from local machine ? Yes, it’s possible and this tutorial demonstrates it with exhaustive examples.

SSH

Practically every Unix and Linux system includes the ssh command. This command is used to start the SSH client program that enables secure connection to the SSH server on a remote machine. The ssh command is used from logging into the remote machine, transferring files between the two machines, and for executing commands on the remote machine.The ssh command provides a secure encrypted connection between two hosts over an insecure network. This connection can also be used for terminal access, file transfers, and for tunneling other applications. Graphical X11 applications can also be run securely over SSH from a remote location.

Other SSH Commands

There are other SSH commands besides the client ssh. Each has its own page. ● ssh-keygen - creates a key pair for public key authentication
● ssh-copy-id - configures a public key as authorized on a server
● ssh-agent - agent to hold private key for single sign-on
● ssh-add - tool to add a key to the agent

● scp - file transfer client with RCP-like command interface
● sftp - file transfer client with FTP-like command interface
● sshd - OpenSSH server

Using the Linux client

Linux typically uses the OpenSSH client. The ssh command to log into a remote machine is very simple. To log in to a remote computer called **sample.ssh.com** , type the following command at a shell prompt:ssh sample.ssh.com
If this is the first time you use ssh to connect to this remote machine, you will see a message like:
The authenticity of host 'sample.ssh.com' cannot be established.
DSA key fingerprint is 04:48:30:31:b0:f3:5a:9b:01:9d:b3:a7:38:e2:b1:0c.
Are you sure you want to continue connecting (yes/no)?
Type yes to continue. This will add the server to your list of known hosts (~/.ssh/known_hosts) as seen in the following message:
Warning: Permanently added 'sample.ssh.com' (DSA) to the list of known hosts. Each server has a host key , and the above question related to verifying and saving the host key, so that next time you connect to the server, it can verify that it actually is the same server. Once the server connection has been established, the user is authenticated. Typically, it asks for a password. For some servers, you may be required to type in a one-time password generated by a special hardware token.
Once authentication has been accepted, you will be at the shell prompt for the remote machine.

Specifying a different user name

It is also possible to use a different username at the remote machine by entering the command

as:
ssh alternative-username@sample.ssh.com
The above can also be expressed with the syntax:
ssh -l alternative-username sample.ssh.com

Executing remote commands on the server

The ssh command is often also used to remotely execute commands on the remote machine without logging in to a shell prompt. The syntax for this is:
ssh hostname command
For example, to execute the command:
ls /tmp/doc
on host **sample.ssh.com** , type the following command at a shell prompt: ssh sample.ssh.com ls /tmp/doc
After authenticating to the remote server, the contents of the remote directory will be displayed, and you will return to your local shell prompt. **-x** Disables X11 forwarding.

SSH client configuration file

The ssh command reads its configuration from the SSH client configuration file ~/.ssh/config. For more information, see the page on SSH client configuration file .

Configuring public key authentication

To configure passwordless public key authentication , you may want to create an SSH key and set up an authorized_keys file. See the pages on ssh-keygen and ssh-copy-id for more information.

Configuring port forwarding

Command-line options can be used to set up port forwarding. Local fowarding means that a local port (at the client computer) is tunneled to an IP address and port from the server. Remote forwarding means that a remote port (at the server computer) is forwarded to a given IP address and port from the client machine. See the page on configuring port forwarding on how to configure them.

OpenSSH also supports forwarding Unix domain sockets and IP packets from a tunnel device to establish a VPN (Virtual Private Network).

SSH command line options

Some of the most important command-line options for the OpenSSH client are: **-1** Use protocol version 1 only.

-2 Use protocol version 2 only.

-4 Use IPv4 addresses only.

-6 Use IPv6 addresses only.

-A Enable forwarding of the authentication agent connection.

-a Disable forwarding of the authentication agent connection.

-C Use data compression

-c cipher_spec Selects the cipher specification for encrypting the session.

-D [bind_address:]port Dynamic application-level port forwarding. This allocates a socket to listen to port on the local side. When a connection is made to this port, the connection is forwarded over the secure channel, and the application protocol is then used to determine where to connect to from the remote machine.

-E log_file Append debug logs to log_file instead of standard error.

-F configfile Specifies a per-user configuration file. The default for the per-user configuration

file is ~/.ssh/config.

-g Allows remote hosts to connect to local forwarded ports.

-i identity_file A file from which the identity key (private key) for public key authentication is read.

-J [user@]host[:port] Connect to the target host by first making a ssh connection to the pjump host[(/iam/jump-host) and then establishing a TCP forwarding to the ultimate destination from there.

-l login_name Specifies the user to log in as on the remote machine.

-p port Port to connect to on the remote host.

-q Quiet mode.

-V Display the version number.

-v Verbose mode.

-X Enables X11 forwarding.

Ssh-keygen

Ssh-keygen is a tool for creating new authentication key pairs for SSH. Such key pairs are used for automating logins, single sign-on, and for authenticating hosts.

SSH Keys and Public Key Authentication

The SSH protocol uses public key cryptography for authenticating hosts and users. The authentication keys, called SSH keys , are created using the keygen program. SSH introduced public key authentication as a more secure alternative to the older .rhosts authentication. It improved security by avoiding the need to have password stored in files, and eliminated the possibility of a compromised server stealing the user's password. However, SSH keys are authentication credentials just like passwords. Thus, they must be managed somewhat analogously to user names and passwords. They should have a proper termination process so that keys are removed when no longer needed.

Creating an SSH Key Pair for User Authentication

The simplest way to generate a key pair is to run ssh-keygen without arguments. In this case, it will prompt for the file in which to store keys. Here's an example:

```
klar (11:39) ~>ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ylo/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ylo/.ssh/id_rsa.
Your public key has been saved in /home/ylo/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:Up6KjbnEV4Hgfo75YM393QdQsK3Z0aTNBz0DoirrW+c
ylo@klar The key's randomart image is:
+---[RSA 2048]----+
 | . ..oo..|
 | . . . . .o.X.|
 | . . o. ..+ B|
 | . o.o .+ ..|
 | ..o.S o.. |
 | . %o= . |
 | @.B... . |
```

```
 | o.=. o. . . .|
 | .oo E. . .. |
+----[SHA256]-----+
klar (11:40) ~>
```

First, the tool asked where to save the file. SSH keys for user authentication are usually stored in the user's .ssh directory under the home directory. However, in enterprise environments, the location is often different. The default key file name depends on the algorithm, in this case id_rsa when using the default RSA algorithm. It could also be, for example, id_dsa or id_ecdsa. Then it asks to enter a passphrase . The passphrase is used for encrypting the key, so that it cannot be used even if someone obtains the private key file. The passphrase should be cryptographically strong. Our online random password generator is one possible tool for generating strong passphrases.

Choosing an Algorithm and Key Size

SSH supports several public key algorithms for authentication keys. These include: ● rsa - an old algorithm based on the difficulty of factoring large numbers. A key size of at least 2048 bits is recommended for RSA; 4096 bits is better. RSA is getting old and significant advances are being made in factoring. Choosing a different algorithm may be advisable. It is quite possible the RSA algorithm will become practically breakable in the

foreseeable future. All SSH clients support this algorithm.
● dsa - an old US government Digital Signature Algorithm. It is based on the difficulty of computing discrete logarithms. A key size of 1024 would normally be used with it. DSA in its original form is no longer recommended.
● ecdsa - a new Digital Signature Algorithm standarized by the US government, using elliptic curves. This is probably a good algorithm for current applications. Only three key sizes are supported: 256, 384, and 521 (sic!) bits. We would recommend always using it with 521 bits, since the keys are still small and probably more secure than the smaller keys (even though they should be safe as well). Most SSH clients now support this algorithm.
● ed25519 - this is a new algorithm added in OpenSSH. Support for it in clients is not yet universal. Thus its use in general purpose applications may not yet be advisable. The algorithm is selected using the -t option and key size using the -b option. The following commands illustrate:

```
ssh-keygen -t rsa -b 4096
ssh-keygen -t dsa
ssh-keygen -t ecdsa -b 521
ssh-keygen -t ed25519
```

Specifying the File Name

Normally, the tool prompts for the file in which to store the key. However, it can also be specified on the command line using the -f <filename> option.

```
ssh-keygen -f ~/tatu-key-ecdsa -t ecdsa -b 521
```

Copying the Public Key to the Server

To use public key authentication, the public key must be copied to a server and installed in an authorized_keys file. This can be conveniently done using the ssh-copy-id tool. Like this:

```
ssh-copy-id -i ~/.ssh/tatu-key-ecdsa user@host
```

Once the public key has been configured on the server, the server will allow any connecting user that has the private key to log in. During the login process, the client proves possession of the private key by digitally signing the key exchange.

Adding the Key to SSH Agent

ssh-agent is a program that can hold a user's private key, so that the private key passphrase only needs to be supplied once. A connection to the agent can also be forwarded when logging into a server, allowing SSH commands on the server to use the agent running on the user's desktop.
For more information on using and configuring the SSH agent, see the ssh-agent page.

Creating Host Keys

The tool is also used for creating host authentication keys. Host keys are stored in the /etc/ssh/ directory.
Host keys are just ordinary SSH key pairs. Each host can have one host key for each algorithm. The host keys are almost always stored in the following files:

```
/etc/ssh/ssh_host_dsa_key
/etc/ssh/ssh_host_ecdsa_key
/etc/ssh/ssh_host_ed25519_key
/etc/ssh/ssh_host_rsa_key
```

The host keys are usually automatically generated when an SSH server is installed. They can be regenerated at any time. However, if host keys are changed, clients may warn about changed keys. Changed keys are also reported when someone tries to perform a man-in-the-middle attack. Thus it is not advisable to train your users to blindly accept them. Changing the keys is thus either best done using an SSH key management tool that also changes them on clients, or using certificates.

Using X.509 Certificates for Host Authentication

OpenSSH does not support X.509 certificates. Tectia SSH does support them. X.509 certificates are widely used in larger organizations for making it easy to change host keys on a period basis while avoiding unnecessary warnings from clients. They also allow using strict host key checking, which means that the clients will outright refuse a connection if the host key has changed.

Using OpenSSH's Proprietary Certificates

OpenSSH has its own proprietary certificate format, which can be used for signing host certificates or user certificates. For user authentication, the lack of highly secure certificate authorities combined with the inability to audit who can access a server by inspecting the server makes us recommend against using OpenSSH certificates for user authentication. However, OpenSSH certificates can be very useful for server authentication and can achieve similar benefits as the standard X.509 certificates. However, they need their own infrastructure for certificate issuance. See more information on certificate authentication .

Key Management Requires Attention

It is easy to create and configure new SSH keys. In the default configuration, OpenSSH allows any user to configure new keys. The keys are permanent access credentials that remain valid even after the user's account has been deleted.In organizations with more than a few dozen users, SSH keys easily accumulate on servers and service accounts over the years. We have
seen enterprises with several million keys granting access to their production servers. It only takes one leaked, stolen, or misconfigured key to gain access.In any larger organization, use of SSH key management solutions is almost necessary. SSH keys should also be moved to root-owned locations with proper provisioning and termination processes. For more information, see how to manage SSH keys. A widely used SSH key management tool for OpenSSH is

Universal SSH Key Manager .Practically all cybersecurity regulatory frameworks require managing who can access what. SSH keys grant access, and fall under this requirement. This, organizations under compliance mandates are required to implement proper management processes for the keys. NIST IR 7966 is a good starting point.

Make Sure There Is Enough Randomness

It is important to ensure there is enough unpredictable entropy in the system when SSH keys are generated. There have been incidents when thousands of devices on the Internet have shared the same host key when they were improperly configured to generate the key without proper randomness.

General Purpose Systems

On general purpose computers, randomness for SSH key generation is usually not a problem. It may be something of an issue when initially installing the SSH server and generating host keys, and only people building new Linux distributions or SSH installation packages generally

need to
worry about it.Our recommendation is to collect randomness during the whole installation of the operating system, save that randomness in a random seed file. Then boot the system, collect some more randomness during the boot, mix in the saved randomness from the seed file, and only then generate the host keys. This maximizes the use of the available randomness. And make sure the random seed file is periodically updated, in particular make sure that it is updated after generating the SSH host keys.Many modern general-purpose CPUs also have hardware random number generators. This helps a lot with this problem. The best practice is to collect some entropy in other ways, still keep it in a random seed file, and mix in some entropy from the hardware random number generator. This way, even if one of them is compromised somehow, the other source of randomness should keep the keys secure.

Embedded Devices and Internet of Things

Available entropy can be a real problem on small IoT devices that don't have much other activity on the system. They may just not have the mechanical randomness from disk drive mechanical movement timings, user-caused interrupts, or network traffic. Furthermore, embedded devices often run on low-end processors that may not have a hardware random number generator.The availability of entropy is also critically important when such devices generate keys for HTTPS.Our recommendation is that such devices should have a hardware random number generator. If the CPU does not have one, it should be built onto the motherboard. The cost is rather small.

Command and Option Summary

Here's a summary of commonly used options to the keygen tool:
-b “Bits” This option specifies the number of bits in the key. The regulations that govern the use case for SSH may require a specific key length to be used. In general, 2048 bits is considered to be sufficient for RSA keys.
-e “Export” This option allows reformatting of existing keys between the OpenSSH key file format and the format documented in RFC 4716 , “SSH Public Key File Format”. **-p** “Change the passphrase” This option allows changing the passphrase of a private key file with **[-P old_passphrase]** and **[-N new_passphrase]** , **[-f keyfile]** .
-t “Type” This option specifies the type of key to be created. Commonly used values are:
- rsa for RSA keys **- dsa** for DSA keys **- ecdsa** for elliptic curve DSA keys
-i "Input" When *ssh-keygen* is required to access an existing key, this option designates the file. **-f** "File" Specifies name of the file in which to store the created key.
-N "New" Provides a new passphrase for the key.

-P "Passphrase" Provides the (old) passphrase when reading a key.
-c "Comment" Changes the comment for a keyfile.
-p Change the passphrase of a private key file.
-q Silence ssh-keygen.
-v Verbose mode.
-l "Fingerprint" Print the fingerprint of the specified public key.
-B "Bubble babble" Shows a "bubble babble" (Tectia format) fingerprint of a keyfile. **-F** Search for a specified hostname in a known_hosts file.
-R Remove all keys belonging to a hostname from a known_hosts file.
-y Read a private OpenSSH format file and print an OpenSSH public key to stdout. This only listed the most commonly used options. For full usage, including the more exotic and

special-purpose options, use the man ssh-keygen command.

ssh-copy-id

ssh-copy-id installs an SSH key on a server as an authorized key. Its purpose is to provision access without requiring a password for each login. This facilitates automated, passwordless logins and single sign-on using the SSH protocol. The ssh-copy-id tool is part of OpenSSH .

Setting up public key authentication

Key based authentication in SSH is called public key authentication . The purpose of ssh-copy-id is to make setting up public key authentication easier. The process is as follows.

Generate an SSH Key

With OpenSSH , an SSH key is created using ssh-keygen . In the simplest form, just run ssh-keygen and answer the questions. The following example illustates this.

```
# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ylo/.ssh/id_rsa): mykey
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in mykey.
Your public key has been saved in mykey.pub.
The key fingerprint is:
SHA256:GKW7yzA1J1qkr1Cr9MhUwAbHbF2NrIPEgZXeOUOz3Us
ylo@klar The key's randomart image is:
+---[RSA 2048]----+
 |.*++ o.o. |
 |.+B + oo. |
 | +++ *+. |
 | .o.Oo.+E |
 | ++B.S. |
 | o * =. |
 | + = o |
 | + = = . |
 | + o o |
+----[SHA256]-----+
#
```

Creating a key pair (public key and private key) only takes a minute. The key files are usually stored in the ~/.ssh directory.

Copy the key to a server

Once an SSH key has been created, the ssh-copy-id command can be used to install it as an authorized key on the server. Once the key has been authorized for SSH, it grants access to the server without a password.

Use a command like the following to copy SSH key:

```
ssh-copy-id -i ~/.ssh/mykey user@host
```

This logs into the server host, and copies keys to the server, and configures them to grant access by adding them to the authorized_keys file. The copying may ask for a password or other authentication for the server.

Only the public key is copied to the server. The private key should never be copied to another machine.

Test the new key

Once the key has been copied, it is best to test it:

```
ssh -i ~/.ssh/mykey user@host
```

The login should now complete without asking for a password. Note, however, that the command might ask for the passphrase you specified for the key.

Troubleshooting

There are a number of reasons why the test might fail:

- The server might not be configured to accept public key authentication. Make sure /etc/ssh/sshd_config on the server contains PubkeyAuthentication yes. Remember to restart the sshd process on the server.
- If trying to login as root , the server might not be configured to allow root logins. Make sure /etc/sshd_config includes PermitRootLogin yes, PermitRootLogin prohibit-password, or without-password. If it is set to forced-commands-only, the key must be manually configured to use a forced command (see command= option in ~/.ssh/authorized_keys.
- Make sure the client allows public key authentication. Check that /etc/ssh/config includes PubkeyAuthentication yes.
- Try adding -v option to the ssh command used for the test. Read the output to see what it says about whether the key is tried and what authentication methods the server is willing to accept.
- OpenSSH only allows a maximum of five keys to be tried authomatically. If you have more keys, you must specify which key to use using the -i option to ssh.

How ssh-copy-id works

ssh-copy-id uses the SSH protocol to connect to the target host and upload the SSH user key. The command edits the authorized_keys file on the server. It creates the .ssh directory if it doesn't exist. It creates the authorized keys file if it doesn't exist. Effectively, ssh key copied to server. It also checks if the key already exists on the server. Unless the -f option is given, each key is only added to the authorized keys file once.It further ensures that the key files have appropriate permissions. Generally, the user's home directory or any file or directory containing keys files should not be writable by anyone else. Otherwise someone else could add new authorized keys for the user and gain access. Private key files should not be readable by anyone else.

Some best practices for SSH keys

SSH keys are very useful, but can lead to problems if they are not properly managed. They are access credentials just like user names and passwords. If they are not properly removed when

people leave or systems are decommissioned, no-one may any longer know who really has access to which systems and data. Many large organizations have ended up having millions of SSH keys.

Use a passphrase when possible

It is recommended that keys used for single sign-on have a passphrase to prevent use of the key if it is stolen or inadvertatly leaked. The ssh-agent and ssh-add programs can be used to avoid having to enter the passphrase every time the key is used.Generally all keys used for interactive access should have a passphrase. Keys without a passphrase are useful for fully automated processes. They allow shell scripts, programs, and management tools to log into servers unattended. This is often used for backups and data transfers between information systems.

Add a command restriction when possible

The copy-id tool does not automatically add command restrictions to keys. Using command restrictions is highly recommended when the key is used for automating operations, such as running a report for fetching some files. A command restriction is basically a command="<permitted command>" option added to the beginning of the line in the server's authorized_keys file.

Managing SSH keys

Anyone having more than a few dozen servers is strongly recommended to manage SSH keys . Not managing the keys exposes the organization to substantial risks, including loss of confidentiality, insertion of fraudulent transactions, and outright destruction of systems.The copy-id tool can be dangerous. It can easily accidentally install multiple keys or unintended keys as authorized. The logic for choosing which key to install is convoluted. Extra authorized keys grant permanent access. They can later be used to spread attacks host-to-host, and the more keys there are, the higher the risk. It also violates all regulatory compliance requirements . The Universal SSH Key Manager is a widely used product for managing SSH keys.

Command-line options

The sample below presents *ssh-copy-id* command line syntax:
ssh-copy-id [-f] [-n] [-i identity file] [-p port] [-o ssh_option] [user@]hostname The options have the following meaning:
-f Don't check if the key is already configured as an authorized key on the server. Just add it. This can result in multiple copies of the key in authorized_keys files.
-i Specifies the identity file that is to be copied (default is ~/.ssh/id_rsa). If this option is not provided, this adds all keys listed by ssh-add -L. Note: it can be multiple keys and **adding extra authorized keys can easily happen accidentally** ! If ssh-add -L returns no keys, then the most recently modified key matching ~/.ssh/id*.pub, excluding those matching ~/.ssh/*-cert.pub, will be used.
-n Just print the key(s) that would be installed, without actually installing them. **-o ssh_option** Pass -o ssh_option to the SSH client when making the connection. This can be used for overriding configuration settings for the client. See ssh command line options and the possible configuration options in ssh_config .
-p port Connect to the specifed SSH port on the server, instead of the default port 22. **-h** or **-?** Print usage summary.

Ssh-copy-id on Mac

While MacOS includes SSH, it does not include ssh-copy-id out of the port. However, according to some sources MacOS 10.12.4 includes it, and presumably newever versions include it as

well.You can test whether your Mac has it by opening a terminal window (Finder / Go / Utilities / Terminal) and typing ssh-copy-id.If your system does not have it, there are many ways to install ssh-copy-id Mac version.

Installation using Homebrew

To install it using Homebrew, use the following command. You need to have the brew command installed. brew install ssh-copy-id

Installation from MacPorts

The following command will install it using MacPorts. You need to have the port command installed. sudo port install openssh +ssh-copy-id

Installation using Curl

The following command can be used to install a Mac version directly. Note that as a general rule we do not recommend piping any commands from the network to the shell, like this does. Only use this method if you fully trust the source. The advantage of this method is that it does not need any special software - curl comes preinstalled.

curl -L https://raw.githubusercontent.com/beautifulcode/ssh-copy-id-for-OSX/master/install.sh | sh

Scp

SCP or secure copy allows secure transferring of files between a local host and a remote host or between two remote hosts. It uses the same authentication and security as the Secure Shell (SSH) protocol from which it is based. SCP is loved for it’s simplicity, security and pre-installed availability.

SCP examples

● Copy file from a remote host to local host SCP example:

```
$ scp username@from_host:file.txt /local/directory/
```

● Copy file from local host to a remote host SCP example:

```
$ scp file.txt username@to_host:/remote/directory/
```

● Copy directory from a remote host to local host SCP example:

```
$ scp -r username@from_host:/remote/directory/ /local/directory/
```

● Copy directory from local host to a remote hos SCP example:

```
$ scp -r /local/directory/ username@to_host:/remote/directory/
```

● Copy file from remote host to remote host SCP example:

```
$ scp username@from_host:/remote/directory/file.txt
username@to_host:/remote/directory/
```

Notes:

— SCP example: scp -r root@123.123.123.123:/var/www/html/ /home/hydn/backups/test/ Also see: Backup solutions .

— Host can be IP or domain name. Once you click return, you will be prompted for SSH password.

— Although this page covers SCP Linux, the instructions will also work for Mac using “Terminal”. You can also use WinSCP to accomplish this on a Windows PC/server. — When copying a source file to a target file which already exists, scp will replace the contents of the target file. So be careful.

SCP options:

– **r** Recursively copy entire directories. Note that this follows symbolic links encountered in the tree traversal.

-C Compression enable. Passes the -C flag to ssh to enable compression.

-l limit – Limits the used bandwidth, specified in Kbit/s.

-o ssh_option – Can be used to pass options to ssh in the format used in ssh_config. **-P** port – Specifies the port to connect to on the remote host. Note that this option is written with a capital ‘P’.

-p Preserves modification times, access times, and modes from the original file. **-q** Quiet mode: disables the progress meter as well as warning and diagnostic messages from ssh.
-v Verbose mode. Print debugging messages about progress. This is helpful in debugging connection, authentication, and configuration problems.

Chapter 6

Scheduling

## cron

cron is a tool for configuring scheduled tasks on Unix systems. It is used to schedule commands or scripts to run periodically and at fixed intervals. Tasks range from backing up the user's home folders every day at midnight, to logging CPU information every hour.
For instance, you can automate process like **backup** , **schedule updates** and **synchronization of files** and many more. **Cron** is a daemon to run schedule tasks. Cron wakes up every minute and checks schedule tasks in crontable. **Crontab** (**CRON TABle**) is a table where we can schedule such kind of repeated tasks.
Tips: Each user can have their own crontab to create, modify and delete tasks. By default **cron** is enable to users, however we can restrict adding entry in **/etc/cron.deny** file.

Cron Command Examples in Linux

Crontab file consists of command per line and have six fields actually and separated either of space or tab. The beginning five fields represent time to run tasks and last field is for command. 1. Minute (hold values between **0-59**)
2. Hour (hold values between **0-23**)
3. Day of Month (hold values between **1-31**)
4. Month of the year (hold values between **1-12** or **Jan-Dec** , you can use first three letters of each month's name i.e **Jan or Jun** .)
5. Day of week (hold values between **0-6** or **Sun-Sat** , Here also you can use first three letters of each day's name i.e **Sun or Wed** .)
6. Command

1. List Crontab Entries

List or manage the task with crontab command with **-l** option for current user. **# crontab -l**

```
00 10 * * * /bin/ls >/ls.txt
```

2. Edit Crontab Entries

To edit crontab entry, use **-e** option as shown below. In the below example will open schedule jobs in **VI** editor. Make a necessary changes and quit pressing **:wq** keys which saves the setting automatically.

# crontab -e

3. List Scheduled Cron Jobs

To list scheduled jobs of a particular user called **tecmint** using option as **-u** (**User**) and **-l** (**List**). **# crontab -u tecmint -l**

```
no crontab for tecmint
```

Note: Only **root** user have complete privileges to see other users crontab entry. Normal user can't view it others.

4. Remove Crontab Entry

Caution: Crontab with **-r** parameter will remove complete scheduled jobs without confirmation from crontab. Use **-i** option before deleting user's crontab.

# crontab -r

5. Prompt Before Deleting Crontab

crontab with **-i** option will prompt you confirmation from user before deleting user's crontab.

# crontab -i -r

crontab: really delete root's crontab?

6. Allowed special character (*, -, /, ?, #)

1. **Asterik(*)** – Match all values in the field or any possible value.
2. **Hyphen(-)** – To define range.
3. **Slash (/)** – 1st field /10 meaning every ten minute or increment of
range. 4. **Comma (,)** – To separate items.

7. System Wide Cron Schedule

System administrator can use predefine cron directory as shown
below. 1. /etc/cron.d
2. /etc/cron.daily
3. /etc/cron.hourly
4. /etc/cron.monthly
5. /etc/cron.weekly

8. Schedule a Jobs for Specific Time

The below jobs delete empty files and directory from **/tmp** at **12:30** am daily. You need to mention user name to perform crontab command. In below example **root** user is performing cron job.

# crontab -e

```
30 0 * * * root find /tmp -type f -empty -delete
```

9. Special Strings for Common Schedule

Strings Meanings

@reboot	Command will run when the system reboot.
@daily	Once per day or may use @midnight.
@weekly	Once per week.
@yearly	Once per year. we can use @annually keyword also.

Need to replace five fields of cron command with keyword if you want to use the
same. **10. Multiple Commands with Double amper-sand(&&)**

In below example command1 and command2 run daily.

# crontab -e

```
@daily <command1> && <command2>
```

11. Disable Email Notification.

By default cron send mail to user account executing cronjob. If you want to disable it add your cron job similar to below example. Using **>/dev/null 2>&1** option at the end of the file will redirect all the output of the cron results under **/dev/null** .

```
[root@tecmint ~]# crontab -e
* * * * * >/dev/null 2>&1
```

At

The **at** command schedules a command to be run once at a particular time that you normally have permission to run. The at command can be anything from a simple reminder message, to a complex script. You start by running the **at** command at the command line, passing it the scheduled time as the option. It then places you at a special prompt, where you can type in the command (or series of commands) to be run at the scheduled time. When you're done, press **Control-D** on a new line, and your command will be placed in the queue. A typical **at** command sequence looks like this (commands you type are shown here in the blue box, or in bold face below):

```
at 9:30 PM Tue
```

```
warning: commands will be executed using /bin/sh
at> echo "It's 9:30 PM on Sunday."
at> ^D
job 1 at Sun Nov 16 09:30:00 2014
```

When we ran the command, the first thing **at** did was give us a "warning" telling us what command shell our commands will be run with: in this case, **/bin/sh** , the Bourne Shell . This shell is the traditional standard Unix shell.

It then places us at the **at>** prompt. Here we type in a simple **echo** command, which echoes a string of text. We press enter, and we're placed at a new **at>** prompt. We then press **Control-D** , telling **at** we're all done with our commands. It then tells us that our job is job number **1** and that it will run next Tuesday.

Specifying Time

at uses a very casual representation of time and date. It even knows some "commonly used" times you might not expect — it knows that "teatime" is traditionally at 4 PM, for instance. Here are some examples of times you can pass to **at** to schedule a command. For instance, let's assume the current time is 10:00 AM, Tuesday, October 18, 2014. The following expressions would translate to the following times:

the expression:	**would translate to:**
noon	12:00 PM October 18 2014
midnight	12:00 AM October 19 2014
teatime	4:00 PM October 18 2014
tomorrow	10:00 AM October 19 2014
noon tomorrow	12:00 PM October 19 2014
next week	10:00 AM October 25 2014
next monday	10:00 AM October 24 2014
fri	10:00 AM October 21 2014
NOV	10:00 AM November 18 2014
9:00 AM	9:00 AM October 19 2014
2:30 PM	2:30 PM October 18 2014
1430	2:30 PM October 18 2014
2:30 PM tomorrow	2:30 PM October 19 2014
2:30 PM next month	2:30 PM November 18 2014
2:30 PM Fri	2:30 PM October 21 2014
2:30 PM 10/21	2:30 PM October 21 2014
2:30 PM Oct 21	2:30 PM October 21 2014
2:30 PM 10/21/2014	2:30 PM October 21 2014
2:30 PM 21.10.14	2:30 PM October 21 2014
now + 30 minutes	10:30 AM October 18 2014
now + 1 hour	11:00 AM October 18 2014
now + 2 days	10:00 AM October 20 2014
4 PM + 2 days	4:00 PM October 20 2014
now + 3 weeks	10:00 AM November 8 2014
now + 4 months	10:00 AM February 18 2015

```
now + 5 years 10:00 AM October 18 2019
```

...so if you run the command:

```
at now + 10 years
```

...and then enter a command at the **at>** prompt, press enter, and type **Control-D** , you will be

mailed the results of your command ten years from now.

Note

If you don't specify a time at the command line, **at** will return the following error message: Garbled time

...and no job will be added to the queue. So, always specify your time at the command line.

Using atq To View Your at Queue

You can use the program **atq** to view your currently-queued **at** jobs. Type **atq** to display the queue.

```
atq
1 Fri Oct 22 09:48:00 2014 a hope
```

This information is, from left to right: **job number** , **date** , **hour** , **year** , **queue** , and **username** . **atq** will only list jobs that belong to you — unless you are the super user, in which case it will list the jobs of all users. So to list all **at** jobs currently queued on the system, type this command (if you have superuser privileges):

```
sudo atq
```

...and type your password, when prompted.

Syntax

at [-V] [-q *queue*] [-f *file*] [-mMlv] *timespec* ...
at [-V] [-q *queue*] [-f *file*] [-mMkv] [-t *time*]
at -c *job* [*job* ...]
atq [-V] [-q *queue*]
at [-rd] *job* [*job* ...]
atrm [-V] *job* [*job* ...]
batch
at -b

Technical Description

at and **batch** read commands from standard input or a specified file that are to be executed at a later time, using sh .

at executes commands at a specified time.

atq lists the user's pending jobs, unless the user is the superuser ; in that case, everybody's jobs are listed. The format of the output lines (one for each job) is: **job number** , **date** , **hour** , **year** , **queue** , and **username** .

atrm deletes jobs, identified by their job number.

batch executes commands when system load levels permit; in other words, when the load average drops below 1.5, or the value specified in the invocation of **atd** .

At allows fairly complex time specifications, extending the POSIX.2 standard. It accepts times of the form **HH:MM** to run a job at a specific time of day. (If that time is already past, the next day is assumed.) You may also specify **midnight** , **noon** , or **teatime** (4pm) and you can have a time-of-day suffixed with AM or PM for running in the morning or the evening. You can also say what day the job will be run, by giving a date in the form month-name day with an optional

year, or giving a date of the form **MMDD[CC]YY** , **MM/DD/[CC]YY** , **DD.MM.[CC]YY** or **[CC]YY-MM-DD** . The specification of a date must follow the specification of the time of day. You can also give times like **now + count** *time-units* , where the *time-units* can be minutes, hours, days, or weeks and you can tell at to run the job today by suffixing the time with today and to run the job tomorrow by suffixing the time with tomorrow.
For example, to run a job at 4pm three days from now, you would do at **4pm + 3 days** , to run a

job at 10:00am on July 31, you would do at **10am Jul 31** and to run a job at 1am tomorrow, you would do at **1am tomorrow** .
The definition of the time specification can be found in **/usr/share/doc/at/timespec** . For both **at** and **batch** , commands are read from standard input or the file specified with the **-f** option and executed. The working directory , the environment (except for the variables **BASH_VERSINFO** , **DISPLAY** , **EUID** , **GROUPS** , **SHELLOPTS** , **TERM** , **UID** , and **_**) and the umask are retained from the time of invocation.
As at is currently implemented as a setuid program, other environment variables (e.g., **LD_LIBRARY_PATH** or **LD_PRELOAD**) are also not exported. This may change in the future. As a workaround, set these variables explicitly in your job.
An **at** or **batch** command run from a **su** shell will retain the current userid. The user will be mailed standard error and standard output from his commands, if any. Mail will be sent using the command **/usr/sbin/sendmail** . If at is executed from a **su** shell, the owner of the login shell will receive the mail.
The superuser may always use these commands. For other users, permission to use **at** is determined by the files **/etc/at.allow** and **/etc/at.deny** . See **at.allow** for details.

Options

-V Prints the version number to standard error and exits successfully.
-q
queue
Uses the specified queue . A queue designation consists of a single letter; valid queue designations range from **a** to **z** and **A** to **Z** . The **a** queue is the default for **at** and the **b** queue for batch. Queues with higher letters run with increased niceness. The special queue " **=** " is reserved for jobs that are currently running.
If a job is submitted to a queue designated with an uppercase letter, the job is treated as if it were submitted to batch at the time of the job. Once the time is reached, the batch processing rules with respect to load average apply. If **atq** is given a specific queue, it will only show jobs pending in that queue.
-m Send mail to the user when the job has completed even if there was no output. **-M** Never send mail to the user. In other words, execute the command, but do not notify the user of its output.
-f *file* Reads the job from *file* rather than standard input.
-t *time* Run the job at *time* , given in the format [[*CC*] *YY*] *MMDDhhmm* [**.** *ss*]. **-l** Running **at -l** is the same as running **atq** ; it displays all queued **at** jobs. **-r** Is the same as running **atrm** . It removes a job from the **at** queue.
-d Is also an alias for **atrm** .
-b Is an alias for **batch** .
-v Shows the time the job will be executed before reading the job. Times displayed will be in the format "Thu Feb 20 14:50:00 1997".
-c The **cats** the jobs listed on the command line to standard output.

Files

/var/spool/cron/atjobs
/var/spool/cron/atspool
/proc/loadavg
/var/run/utmp
/etc/at.allow
/etc/at.deny

Examples

at -m 01:35 < my-at-jobs.txt
Run the commands listed in the ' **my-at-jobs.txt** ' file at **1:35** AM. All output from the job will be mailed to the user running the task. When this command has been successfully entered you should receive a prompt similar to the example below:
commands will be executed using /bin/sh
job 1 at Wed Dec 24 00:22:00 2014
at -l
This command will list each of the scheduled jobs in a format like the following: 1 Wed Dec 24 00:22:00 2003
...this is the same as running the command **atq** .
at -r 1
Deletes job **1** . This command is the same as running the command **atrm 1** . atrm 23
Deletes job 23. This command is the same as running the command **at -r 23** .

SAMPLE QUESTIONS

1. Read a number and print whether it is even or odd.
2. Read 3 marks of a student and nd the average. Display the grade of the student based on the average.
S >= 90%
A < 90%, but >= 80% B
< 80%, but >= 60% P <
80%, but >= 40% F <
40%
3.
Change the home folder of all users whose name start with stud from /home/username to /usr/username. Also change the password of user-name to username123
(e.g., /home/stud25 changes to /usr/stud25 and his/her password changes to stud25123) - (Use for .. in)
4. Read a number and display the multiplication table of the number up to 10 lines. - (Use for((..)))
5. Read a Decimal number. Convert it to Binary and display the result. - (Use while) 6. Setup password less ssh to a nearby system. (Or to a virtual machine, running in your system.)
Note down the IP address of the remote machine or VM. use ssh-keygen to generate a key use ssh-copy-id username@IP to copy the key to remote machine. verify

passwordless login using ssh username@IP

cron & at

7. Create a directory backup, in your home folder.

a. Write a script, backupscript that copies (update only) the contents of dir1 to backup. The script also outputs date command to a le called script.output.

b. Use crontab -e to add the following to cron table to execute back-upscript

c. { at 5 P.M., everyday.

d. { at 12 P.M. and 4 P.M. on every Friday.

e. { at 5 A.M. on 1 st of every month.

f. Use at to execute backupscript at 12 P.M. on 31 st March,2018. Note

(a) The contents of the le script.output, will give the instants when backupscript is executed. It can be used for checking the results.

(b) Use Virtual Machine so that you can modify system time and verify the results quickly.

CHAPTER 7

Start/Stop Services

Linux Bootup and Login Sequence

1. BIOS boot-ups; checks peripherals; finds bootable device.
2. First sector loaded into RAM and executed– GRUB prompt appears. 3. Kernel loaded from sector list, /boot/vmlinuz-4.15.0-xx-generic; places the appropriate initial RAM disk image, initrd, into RAM.
4. Kernel executed; unpacks.
5. Kernel initializes hardware.
6. Kernel mounts root, /, file system, say /dev/sda1.
7. Kernel executes init (Systemd) as PID 1.
8. Systemd initiates the units in the default target. The units start the services.
9. Among the started services, there are agetty programs on each terminal and the login process.
10.login requests user name and password and validates them against /etc/passwd, or NIS maps, or LDAP (depending on PAM settings).
11.login starts a shell.
12.The shell executes the appropriate startup files (.profile for sh, ksh; .bash_profile for BASH; .cshrc and .login for csh and tcsh).
13.The shell prints a prompt and waits for input.

Systemd is the parent of all processes

Systemd is the parent of all processes on the system, it is executed by the kernel and is responsible for starting all other processes; it is the parent of all processes whose natural parents have died and it is responsible for reaping those when they die.

Install package psmisc and run command pstree

```
sudo -s
apt-get install psmisc
pstree
```

Output:

```
systemd─┬─agetty
 ├─cron
 ├─dbus-daemon
 ├─login───bash───sudo───bash───pstree
 ├─networkd-dispat───{networkd-dispat}
 ├─rsyslogd───3*[{rsyslogd}]
 ├─sshd
 ├─systemd───(sd-pam)
 ├─systemd-journal
```

```
 ├─systemd-logind
 ├─systemd-network
 ├─systemd-resolve
```

```
├─systemd-timesyn───{systemd-timesyn}
└─systemd-udevd
```

```
pstree -p -l
```

shows the process tree with their PID in the long (non-truncated) format.
/usr/bin/pstree comes with package psmi

Systemd in services startup.

Administrative commands to start/stop services

Systemd driven services are available in Ubuntu 18.04 and RedHat 7 and newer
Command systemctl talks to systemd. To see all the services registered with systemd, run the following:

```
systemctl list-units --type service
```

To stop rsyslog services, run

```
systemctl stop rsyslog
```

To see the status of the service:

```
systemctl status rsyslog
```

Stop syslog.socket then rsyslog.service:

```
systemctl stop syslog.socket
systemctl stop rsyslog.service
```

To see the status of the service:

```
systemctl status rsyslog
```

To start the service and see its status:

```
systemctl start rsyslog
systemctl status rsyslog
```

Legacy System V services

Used to be available on any Linux distro. Now they are provided within Systemd for backward compatibility. The System V (non-event driven) service startup scripts are located in directory /etc/init.d. For example to stop and then start cups printing service, use the following

```
apt-get install cups
/etc/init.d/cups stop
/etc/init.d/cups start
/etc/init.d/cups status
```

Is the same as

```
service cups stop
service cups start
service cups status
```

System Run Levels (Legacy System V)

This is what used to be on older Linux distros. At bootup, init reads /etc/inittab file and executes all scripts for default run level. If the /etc/inittab is absent, the default run level is 2. On Unix, there are 7 conventional run levels (0,1,2,3,4,5,6). However, 10 levels are possible (0 – 9).

On Ubuntu and Debian,

Runlevel 0 -is halt.
Runlevel 1 -is single-user.
Runlevels 2-5 are multi-user.
Runlevel 6 -is reboot.

On the other Linux systems, such as RedHat,

Runlevel 0 - halt
Runlevel 1 - Single user mode
Runlevel 2 - Multiuser, without NFS (The same as 3, if you do not have networking) Runlevel 3 - Full multiuser mode
Runlevel 4 - unused
Runlevel 5 - X11
Runlevel 6 –reboot

Working with Systemd targets

Viewing the Default Target:

```
systemctl get-default
```

Viewing the Current Target

```
systemctl list-units --type target
```

Changing the Default Target

```
systemctl set-default multi-user.target
```

Reboot. Run commands

```
systemctl get-default
systemctl list-units --type target
```
Set password for root:
```
sudo -s
passwd
```
Changing the Current Target
```
systemctl isolate graphical.target
```
Check the Target units
```
systemctl list-units --type target
```
Changing to Rescue Mode
```
systemctl rescue
```
Check the Target units
```
systemctl list-units --type target
```
Changing to Emergency Mode
```
systemctl emergency
```
Check the Target units
```
systemctl list-units --type target
```
Run command pstree to see the number of processes running.

Adding service to systemd startup

In your home directory (i am assuming that it is hostadmin) create a shell script file, test.sh:

File content:
```
#!/bin/bash
echo current date/time is $(/bin/date) >> /home/hostadm/timecheck.txt
```
Make the script executable
```
chmod 755 /home/hostadm/test.sh
```
In directory /lib/systemd/system, create unit service file, test.service:

File content:
```
[Unit]
Description=test startup
[Service]
Type=simple
ExecStart=/home/hostadm/test.sh
[Install]
WantedBy=default.target
```
Enable service test in systemd:
```
systemctl daemon-reload
systemctl enable test
```
Run commands
```
systemctl start test
systemctl status test
```
Check if file /home/hostadm/timecheck.txt has been updated. Reboot the machine.

Run command
```
systemctl status test
```
Check if file /home/hostadm/timecheck.txt has been updated.

Adding stop script to the unit

In directory /home/hostadm, create another shell script file, stop.sh:

script file:

```
#!/bin/bash
echo STOP >> /home/hostadm/timecheck.txt
```

Make the script executable

```
chmod 755 /home/hostadm/stop.sh
```

In directory /lib/systemd/system, modify the unit service file, test.service by adding two lines in the [Service] block:

test.service:

```
[Unit]
Description=test startup
[Service]
Type=simple
ExecStart=/home/hostadm/test.sh
RemainAfterExit=true
ExecStop=/home/hostadm/stop.sh
[Install]
WantedBy=default.target
```

Reload systemd:

```
systemctl daemon-reload
```

Run commands

```
systemctl stop test
systemctl status test
```

Check if file /home/hostadm/timecheck.txt has been updated with entry “STOP”, generated by the stop.sh script.

User settings at login.

When a user logins, getty gives him SHELL.

In case of /bin/bash, the following initialization/configuration scripts are executed:

/etc/profile System wide initialization file. Runs at log in.

~/.bash_pro
file

Executed automatically after /etc/profile

~/.bash_log
in

If ~/.bash_profile is absent, the script is executed after /etc/profile,
otherwise, ignored.

~/.profile

If both ~/.bash_profile and ~/.bash_login are absent, the script
is executed after /etc/profile, otherwise, ignored.

~/.bashrc
It is called from ~/.bash_profile at startup. An interactive shell that is
not a login shell also reads ~/.bashrc .
~/.bash_log

out
Executed automatically during log out
When you need to specify environment variables, for example PATH, you can place them in /etc/profile for all users on the system, or in ~/.bashrc for the specific user. When the user logs out, control returns to init, which wakes up and spawns a new getty on the terminal port.

Login scripts

In a user home directory, there should be scripts .profile and .bashrc. Create a new user account, for example jerry:

```
useradd -m -s /bin/bash jerry
passwd jerry
```

In file .profile, put entry
file content
echo PROFILE
In file .bashrc, put entry
file content
echo BASHRC
Logout and login as user jerry and notice PROFILE in the login prompt. Start a new shell,

```
/bin/bash
```

notice BASHRC in the new shell prompt.

Chapter 8
Installation of LAMP Stack

1. Update your system

```
sudo apt update
```

2. Install Apache using apt:

```
sudo apt install apache2
```

Confirm that Apache is now running with the following command:

```
sudo systemctl status apache2
```

if it is not working

```
sudo systemctl start apache2
```

Once installed, test by accessing your server’s IP in your browser:

http://youripaddress

(find out your ip address using ifconfig)

3. Install mariadb

(MariaDB is a fork of MySQL from some of the original MySQL team and is a drop-in replacement.)

```
sudo apt install mariadb-server mariadb-client
```

Check mariadb Installation

```
sudo systemctl status mysql
```

(if it is not working sudo systemctl start mysql)

Secure your newly installed MariaDB service:

```
sudo mysql_secure_installation
```

(This will set password for mariadb, and strengthen the security by asking some questions like disallow root login remotely? Remove test database? Etc)

To access the MariaDB shell, run the mysql command with sudo.

sudo mysql **or**

sudo mysql -u root **or**

sudo mysql -u root –p

Create a new user and grant access to the required databases:

```
CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'userpassword';
GRANT ALL PRIVILEGES ON database_name.* TO 'newuser'@'localhost';
```

4. Install PHP and commonly used modules

```
sudo apt install php libapache2-mod-php php-opcache php-cli php-gd
php-curl php-mysql
```

Restart apache2

```
sudo systemctl restart apache2
```

Now you can check php installation

```
sudo echo “<?php phpinfo(); ?>” | sudo tee -a /var/www/html/phpinfo.php
> /dev/null
```

Open a browser

http://127.0.0.1/phpinfo.php

5. Install phpmyadmin

sudo apt install phpmyadmin php-mbstring php-zip php-gd php-json php-curl (It ask for webserver select apache2, select db-configuration and set password)

Restart apache2

```
sudo systemctl restart apache2
```

Configure Apache2 to serve the phpMyAdmin site.

```
sudo ln -s /etc/phpmyadmin/apache.conf /etc/apache2/conf-availabl
e/phpmyadmin.conf
sudo a2enconf phpmyadmin.conf
sudo systemctl reload apache2.service
```

Check phpmyadmin

Open a browser

```
http://localhost/phpmyadmin
username : newuser
password : userpassword
```

Chapter 9
Installing Software From Source Code

1. Compilation stages

Suppose one needs to compile a single source C program, code.c, into an executable binary, code.x:

```
gcc -o code.x code.c
```

This implies the following implicit compilation steps that also can be done in 4 stages:

stage name command product

1 Preprocessing

gcc -E code.c -o

code.i

#define, #ifdef, #include are processed

2 Compiling

gcc -S code.i -o

code.s

Assembly file, code.s
3 Asembling
gcc -c code.s -o
code.o
Creates object file in the machine
language
4 Linking
gcc code.o -o
code.x
Code is linked with the other .o object
files and libraries into an executable
binary

2. Compilation in stages

Please look at the code.c file provided with this tutorial

Look at the content of the source file:

```
less code.c
```

Compile the source file in the four stages - Preprocessing, Compiling, Assembling, and Linking:

```
gcc -E code.c -o code.i
gcc -S code.i -o code.s
gcc -c code.s -o code.o
gcc code.o -o code.x -lm
```

Check the file type of the compilation products:

```
file code.i
```

```
file code.s
file code.o
file code.x
```

Browse the content of the two ASCII files, code.i and code.s:

```
less code.i
less code.s
```

Run the executable, code.x:

```
./code.x
```

3. Source code with multiple files

If a source code consists of several files and needs libraries to be linked with, the compilation procedure can be done either in one step:

```
gcc code.c function.c -o code.x
```

or multiple steps:

```
gcc -c code.c
gcc -c function.c
gcc -o code.x code.o function.o
```

If the header files *.h and the libraries *.a are located in the directories different from the source *.c files, for example ../include and ../lib, their location needs to be specified as follows: (beware of paths)

```
gcc -o code.x code.o function.o extra_2.o -I../include -L../lib -lextra
```

4. Static libraries

A part of the code that can be compiled separately and linked with the other codes should be built as a library.

Building a static library

```
gcc -c funciton.c
ar -crU mylib.a function.o
ranlib mylib.a
```

to see the object files included in the library:

```
ar -t mylib.a
```

or

```
nm mylib.a
```

Comple with our library

```
gcc -o code_stat.x code.o -L. -lmylib
```

5. Shared library

Shared libraries,unlike static ones, are not compiled into the application binaris. They are loaded into the memory and linked with the application at a run time. Other running codes can use the loaded shared libraries.

Building a shared library

```
gcc -c -fPIC function.c
gcc -shared function.o -o libmylib.so
```

Note the name libmylib.so. Lib prefix is required.

To compile the application with the library we would follow the step below: -L . looks in current dir.

```
gcc -o code_shared.x code.c -L. -l mylib
```

(check u may need to copy library to some lib folder , say *usr*local/ib and then run ldconfig) To see what the shared libraries the code needs, use ldd command

```
ldd code.x
```

If the library is not found, include it into LD_LIBRARY_PATH environment variable, for example:

```
export LD_LIBRARY_PATH=.
```

Alternatively the shared library path can be included into /etc/ld.so.conf. After /etc/ld.so.conf has been updated, we need to run command

```
/sbin/ldconfig -p
```

Get the list of shared libraries required by code_stat.x, and code_shared.x:

```
ldd code_stat.x
ldd code_shared.x
```

Compare the sizes of the executable files:

```
ls -lh code_*.x
```

6 Worked Example

Single code compilation and libraries

Look at single.c that computes the scalar product of two vectors: Code is given under folder code_single Move to that folder and try

```
gcc -o app.x single.c
./app.x
```

When prompted for the input parameters, type in the dimension of the vectors and the value of their components, for example:
ouput/input:

```
Vector dimension n=2
Input 2 coordinate x: 3 5
Input 2 coordinate y: 1 6
```

Compile the code with a static library.
Now we will create three separate files from single.c .The main.c contains the main code h file contain header definitons and Scalar_Prodcut.c contains function definiton. These are three files needed
main.c , Scalar_Product.h , and Scalar_Product.c .
The files are available under code_static
Move to that folder and files
Build the static library:

```
gcc -c Scalar_Product.c
ar -cr libScalar_Product.a Scalar_Product.o
ranlib libScalar_Product.a
```

Verify the content of library libScalar_Product.a:

```
ar -t libScalar_Product.a
```

Compile the main code and link with the library:

```
gcc -c main.c
gcc -o app.x main.o -L. -lScalar_Product
```

Make sure the compiled code runs fine:

```
./app.x
```

Compile with shared library
Look at code kept in code_so folder
Build the shared library:

```
gcc -fPIC -c Scalar_Product.c
gcc -shared Scalar_Product.o -o libScalar_Product.so
```

Compile the main code and link with the shared library:

```
gcc -c main.c
gcc -o apps.x main.o -L. -lScalar_Product
```

Verify that apps.x executable requires libScalar_Product.so library:

```
ldd apps.x
```

It should show that the shared library is not found, libScalar_Product.so => not found. Try running the code

```
./apps.x
```

7 Make and Makefile

Utility make is used for compilation and installation of a software with many source files.
make keeps track with the dependencies and file access times tamps.
Compilation is done in the correct order to meet prerequisites and satisfy the dependencies. If one or several source codes have been updated, only

these and the files that depend on them would be recompiled by command make. Makefile example: **Be very careful about the TAB above . Copy paste may bring in error as tabs are sometimes converted to spaces.** Look at the files in make_file folder

Create a file named **makefile** with following content

```
app.x: main.o libScalar_Product.a
gcc -o app.x main.o -L. -lScalar_Product
main.o: main.c Scalar_Product.h
gcc -c main.c
Scalar_Product.o: Scalar_Product.c
gcc -c Scalar_Product.c
libScalar_Product.a: Scalar_Product.o
ar -cru libScalar_Product.a Scalar_Product.o
ranlib libScalar_Product.a
```

The meaning of the above lines can be interpreted as

```
app.x: main.o libScalar_Product.a
gcc -o app.x main.o -L. -lScalar_Product
```

The app.x is rule specifis that when ever ther is a change to main.o or libScalar_Product execute the line below.

```
main.o: main.c Scalar_Product.h
gcc -c main.c
```

When ever there is some change to main.c or Scalar_Product.h recompile main.c to produce main.o . This will in turn trigger the app.x rule as main.o has changed.

The remaining lines implies changes to specific files and their implications to compilation process.

Makefile or makefile would be processed by running command make:

```
make
```

If the Makefile has different name, for example Makefile1, it can be processed as follows:

```
make -f Makefile1
```

8 Makefile with macros

Parameters that appear in the Makefile mutiple times or that need to be modified, depending on the environment, can be expressed through macronames.

Makefile with macros example:

Makefile:

```
#First, we define the MACRONAMES:
SLIB = libScalar_Product.a
APP=app.x
CC = gcc
CFLAGS = -O3
LIBPATH = .
#Then we refer to them by $(MACRONAMES):
$(APP): main.o $(SLIB)
```

```
$(CC) $(CFLAGS) -o $(APP) main.o -L$(LIBPATH) -lScalar_Product
main.o: main.c Scalar_Product.h
$(CC) $(CFLAGS) -c main.c
Scalar_Product.o: Scalar_Product.c
$(CC) $(CFLAGS) -c Scalar_Product.c
$(SLIB): Scalar_Product.o
ar -cru $(SLIB) Scalar_Product.o
ranlib $(SLIB)
```

9 Makefile with phony targets

When a target in a Makefile rather means just an action than a file, it is called a phony target. Makefile with phony targets (clean, install, and uninstall):

Makefile:

```
SLIB = libScalar_Product.a
APP=app.x
CC = gcc
CFLAGS = -O3
LIBPATH = .
INSTPATH = /usr/local
$(APP): main.o $(SLIB)
$(CC) $(CFLAGS) -o $(APP) main.o -L$(LIBPATH) -lScalar_Product
main.o: main.c Scalar_Product.h
$(CC) $(CFLAGS) -c main.c
Scalar_Product.o: Scalar_Product.c
$(CC) $(CFLAGS) -c Scalar_Product.c
$(SLIB): Scalar_Product.o
ar -cru $(SLIB) Scalar_Product.o
ranlib $(SLIB)
clean:
-rm -f *.o *.a $(APP)
install:
@cp -p $(APP) $(INSTPATH)/bin ;\
chown root:root $(INSTPATH)/bin/$(APP) ;\
cp -p $(SLIB) $(INSTPATH)/lib ;\
chown root:root $(INSTPATH)/lib/$(SLIB) ;\
echo "Install $(APP) and $(SLIB) into $(INSTPATH)"
uninstall:
-@rm -f $(INSTPATH)/bin/$(APP)
```