CSNB113: System Administration
-
7th Topic:
Files … Files … Files

"Everything is a file"

… at least in the world of *nix.

A file is a file.

A directory is – kind of – a file (with a 'd' in front of it).

A process is a file (under /proc/).

A device is a file (under /dev).

A link is a file

A mass storage devices (e.g. hard-drive, CD-ROM, USB Key) is a file

Inter-process communication:

- A pipe is a file
- shared memory is a file
- A socket is a file

A Network connection is a file

An interactive terminal is a file

Almost all other devices (e,g, Printers, Graphic Card) are files …

Actually, in the beginning it was: "everything is a stream of Bytes"

Commands for files

ls	gives a list of filenames in the current directory
mv	rename a file: **mv *oldfile newfile*** or **mv *oldfile newdir***
cp	copy a file: **cp *oldfile newfile*** or **cp *oldfile newdir***
chmod	change permissions; see "File Attributes" for examples
rm	remove a file
cd	change directories
lpr	print a file: **lpr -P *printer-name file***
pwd	"print" working directory; returns the current directory

Commands for files

quota	to see how much disk space you've used
mkdir	create a new directory
rmdir	remove a directory
compress	compress a file into one with a **.Z** extension; **uncompress** reverses the process
tar	package a group of files into one file for moving or archiving; also extracts tar files

Archives of files

tar

stands for **t**ape **ar**chive. It was a very early archiving product for *nix. It allows to **create** an archive of many files, including directories. It also **stores the paths** of all files, and it **stores the permissions** of all files.

It allows to **create** an archive, and to **extract** an archive. It also allows to **select files /directories** to be archived and to **select files / directories** to be **extracted** from the archive.

It allows to **compress** an archive with a number of ***compression algorithms*** to make the archive smaller.

Basic tar

-c	create an archive
-v	verbose ("containing many words")
-f	filename
-t	list files in an archive
-x	extract files from an archive
-z	compress / uncompress with gzip
-j	compress / uncompress with bzip2
-p	preserve permissions (default for superuser)
-P	maintains leading '/'-s (Absolute Path)

```
tar -c -v -f demo.tar *.txt
tar -t -v -f demo.tar t*.txt
tar -x -v -f demo.tar t*.txt
```

Basic tar – example I

```
$ ls
a.txt  test  test.txt
$ tar -c -v -f demo.tar *.txt

$ tar -t -v -f demo.tar t*.txt

$ tar -x -v -f demo.tar t*.txt

$
```

Basic tar - example

```
$ tar -c -v -f demo.tar /etc/passwd
tar: Removing leading `/' from member names
/etc/passwd
$ tar -t -v -f demo.tar
-rw-r--r-- root/root      1933 2010-11-25 11:33 etc/passwd
$ tar -c -v -f demo.tar -P /etc/passwd
/etc/passwd
$ tar -x -v -f demo.tar
tar: Removing leading `/' from member names
/etc/passwd
$ ls
a.txt  demo.tar  etc  test  test.txt
$ tar -x -v -f demo.tar -P
/etc/passwd
tar: /etc/passwd: Cannot open: File exists
tar: Exiting with failure status due to previous errors
$
```

Basic tar - example

```
$ tar -c -v -f demo.tar /etc/passwd
tar: Removing leading `/' from member names
/etc/passwd
$ tar -t -v -f demo.tar
-rw-r--r-- root/root      1933 2010-11-25 11:38 etc/passwd
$ tar -c -v -f demo.tar -P /etc/passwd
/etc/passwd
$ tar -x -v -f demo.tar
tar: Removing leading `/' from member names
/etc/passwd
$ ls
a.txt  demo.tar  etc  test  test.txt
$ tar -x -v -f demo.tar -P
/etc/passwd
tar: /etc/passwd: Cannot open: File exists
tar: Exiting with failure status due to previous errors
$
```

tar as backup

tar

is a useful backup tool for files and directories, and very frequently used.

One doesn't want to backup all, all the time, rather depending on the necessity and the changes:

/home containing user files: frequent changes

```
tar -c -v -p -f home.tar /home
```

This can be done regularly, and then, if needed, it can be ***rolled-back***

```
cd /
tar -x -v -p -f home.tar
```

While tar allows to be **very selective** with files and directories, it is prone to mistakes. Often one wants to **back up a whole partition** in a single go, without missing anything, without even a chance to lose anything.

dump

This is the tool to simply **backup a whole partition**. (Another reason to create lots of them at install.)

```
sudo fdisk -l
```

shows us all the partitions on our drive:

```
$ sudo fdisk -l
   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1       29165   234259456   83  Linux
/dev/sda2           29165       30402     9936897    5  Extended
/dev/sda5           29165       30402     9936896   82  Linux swap / Solaris
```

```
dump -0u -f part_1.dmp /dev/sda1
```

creates a file, part_1.dmp, containing all and everything on the first partition one (/dev/sda1). So the whole, complete, partition will be stored as such in a (large) file. If the partition size is 40 GB, but only 2.2 GB are used, the resulting file part_1.dmp will only be 2.2 GB in size.

Still, this is large, and will take some time. Therefore, it is possible to make

-0:a complete backup ("**full backup**")

-1:an **incremental backup**

The first run will make a 1:1 copy of the partition and save it as file; the second run, with option -1, will **only** save those files that have changed since the last, complete, backup

mount – associating files

When you have more than one partition, the behaviour in Windows and *nix is different:

In Windows, you get **another drive**. The first drive, usually C:, contains the system files. If you have a second partition, it will show as D:, and if you add a CDROM, it could show as E: or M:.
Inserting a thumb drive could get you a new drive F:.

In *nix, when a second partition is available, or a CD is inserted, or a thumb drive plugged in, all these file systems need to be **associated** to a location within the existing filesystem; somewhere under '/'.

The term is: It is **associated** with a specific location in the file system hierarchy. This point is called the ***mount point***.

The command for mounting is **`mount`**, for un-mounting it is **`umount`**.

The command without option or parameter displays the currently mounted file systems.

mount - Example

At installation, you created a number of partitions: /boot, /, /home.

When the system is booted, these partitions are automatically mounted: When you log on, you are in your home directory, under /home.

```
$ mount
/dev/sda3 on / …
/dev/sda1 on /boot …
/dev/sda5 on /home …
```

This means, that these partitions are mounted automatically at boot. How does the system know about this? The information about the system partitions are written into file /etc/fstab

vi – ***the*** editor for files

There are plenty of editors, from Windows' Notepad to Office, for GUI as well as command line.

Though one is unique: vi. It is unique, because it is **ubiquitous**.
This means it is available almost everywhere and anywhere on any Linux, Unix, Solaris, *BSD, OSX. But it also is available on most embedded systems.
Even on ADSL-modems and wireless access points one can often find it. It is on routers, firewalls, and almost always on basic installs. It is also unique in the way it handles user input. Actually, it has a very strange way to handle user input. It can be used without mouse, even without cursor keys.
On the other hand, it is very powerful, and some have written whole books with it.

In a nutshell: It has two modes:

1. Command mode. Anything the user types is considered a control / command. There are keys strokes for navigation, up, down, to the end of the line, beginning of the document, replacing some string (words)

2. Input mode. Anything the user types is inserted into the document

One changes from command mode to input mode by pressing 'i' (nsert)

One changes from input to command mode by pressing 'Esc' (ape)

References

- Textbook, p. 691 – 694, Keeping track of your disk space
- Textbook, p. 715 – 729, Working with files
- Textbook, p. 568 – 581
-