5.10 cmp: COMPARING TWO FILES

You may often need to know whether two files are identical so one of them can be deleted. There are three commands in the UNIX system that can tell you that. In this section, we'll have a look at the **cmp** (compare) command. Obviously, it needs two filenames as arguments:

```
$ cmp chap01 chap02
chap01 chap02 differ: char 9, line 1
```

The two files are compared byte by byte, and the location of the first mismatch (in the ninth character of the first line) is echoed to the screen. By default, **cmp** doesn't bother about possible subsequent mismatches but displays a detailed list when used with the `-l` (list) option.

If two files are identical, **cmp** displays no message, but simply returns the prompt. You can try it out with two copies of the same file:

```
$ cmp chap01 chap01
$ _
```

This follows the UNIX tradition of quiet behavior. This behavior is also very important because the comparison has returned a *true* value, which can be subsequently used in a shell script to control the flow of a program.

5.11 comm: WHAT IS COMMON?

Suppose you have two lists of people and you are asked to find out the names available in one and not in the other, or even those common to both. **comm** is the command you need for this work. It requires two *sorted* files, and lists the differing entries in different columns. Let's try it on these two files:

```
$ cat file1
c.k. shukla
chanchal singhvi
s.n. dasgupta
sumit chakrobarty
```

```
$ cat file2
anil aggarwal
barun sengupta
c.k. shukla
lalit chowdury
s.n. dasgupta
```

Both files are sorted and have some differences. When you run **comm**, it displays a three-columnar output:

```
$ comm file[12]                                  Comparing file1 and file2
                anil aggarwal
                barun sengupta
                        c.k. shukla
chanchal singhvi
                lalit chowdury
                        s.n. dasgupta
sumit chakrobarty
```

The first column contains two lines unique to the first file, and the second column shows three lines unique to the second file. The third column displays two lines common (hence its name) to both files.

This output provides a good summary to look at, but is not of much use to other commands that take **comm**'s output as their input. These commands require single-column output from **comm**, and **comm** can produce it using the options -1, -2 or -3. To drop a particular column, simply use its column number as an option prefix. You can also combine options and display only those lines that are common:

```
comm -3 foo1 foo2                    Selects lines not common to both files
comm -13 foo1 foo2                   Selects lines present only in second file
```

The last example and one more with the other matching option (-23) has more practical value than you may think, but we'll not discuss their application in this text.

5.12 diff: CONVERTING ONE FILE TO OTHER

diff is the third command that can be used to display file differences. Unlike its fellow members, **cmp** and **comm**, it also tells you which lines in one file have to be *changed* to make the two files identical. When used with the same files, it produces a detailed output:

```
$ diff file1 file2                   Or diff file[12]
0a1,2                                Append after line 0 of first file
> anil aggarwal                      this line
> barun sengupta                     and this line
2c4                                  Change line 2 of first file
< chanchal singhvi                   Replacing this line
---                                  with
> lalit chowdury                     this line
4d5                                  Delete line 4 of first file
< sumit chakrobarty                  containing this line
```

diff uses certain special symbols and **instructions** to indicate the changes that are required to make two files identical. You should understand these instructions as they are used by the **sed** command, one of the most powerful commands on the system.

Each instruction uses an **address** combined with an **action** that is applied to the first file. The instruction **0a1,2** means appending two lines after line 0, which become lines 1 and 2 in the second file. **2c4** changes line 2 which is line 4 in the second file. **4d5** deletes line 4.

5.14 COMPRESSING AND ARCHIVING FILES

To conserve disk space you'll need to compress large and infrequently used files. Moreover, before sending a large file as an email attachment, it's good etiquette to compress the file first. Every UNIX system comes with some or all of the following compression and decompression utilities:

- **gzip** and **gunzip** (**.gz**)
- **bzip2** and **bunzip2** (**.bz2**)
- **zip** and **unzip** (**.zip**)

You'll find all of these programs on Solaris and Linux. The extension acquired by the compressed filename is shown in parentheses. The degree of compression that can be achieved depends on the type of file, its size and the compression program used. Large text files compress more, but GIF and JPEG image files (the types used on the World Wide Web) compress very little because they hold data in compressed form.

Apart from compressing, you'll also need to group a set of files into a single file, called an **archive**. The **tar** and **zip** commands can pack an entire directory structure into an archive. You can send this archive as a single file, either using **ftp** or as an email attachment, to be used on a remote machine. An additional layer of compression helps bring down the file size, the reason why **tar** is often used with **gzip** and **bzip2** for creating a compressed archive. **zip** handles both functions itself. In the next few sections, we'll be discussing these compression and archival utilities.

Note: Modern versions of WINZIP that run on Windows can read all of these formats though it can write only in the ZIP format. So you can choose any of these formats for compressing and archiving files even if you have to restore them on a Windows system. However, do check the WinZip documentation on the other system before you select the format to write an archive.

5.15 gzip AND gunzip: COMPRESSING AND DECOMPRESSING FILES

We'll monitor the size of one HTML and one PostScript file as they go through a number of compression and archival agents. We'll start with **gzip**, a very popular program, that works with one or more filenames. It provides the extension .gz to the compressed filename and removes the original file.

How well do HTML files compress? In the following example, we run **gzip** on the file libc.html (documentation for the GNU C library). We also note the file size, both before and after compression, using **wc -c** which counts characters:

```
$ wc -c libc.html
  3875302 libc.html
$ gzip libc.html
$ wc -c libc.html.gz
   788096 libc.html.gz
```

We seem to have achieved very high compression here, but before we go in for the statistics, let's repeat the previous exercise on a Postscript file:

```
$ wc -c User_Guide.ps ; gzip User_Guide.ps ; wc -c User_Guide.ps.gz
   372267 User_Guide.ps
   128341 User_Guide.ps.gz
```

Before compression
After compression

How much compression did we actually achieve for both files? Use the -l option with the compressed or original filenames as arguments:

```
$ gzip -l libc.html.gz User_Guide.ps.gz
compressed  uncompr. ratio uncompressed_name
    788096   3875302  79.6% libc.html
    128341    372267  65.5% User_Guide.ps
    916437   4247569  78.4% (totals)
```

.gz *not necessary*

HTML compressed better than Postscript (79.6% vs. 65.5%). This may not always be the case.

5.15.1 gzip Options

Uncompressing a "gzipped" File (-d) To restore the original and uncompressed file, you have two options: Use either **gzip -d** or **gunzip** with one or more filenames as arguments; the .gz extension is optional yet again:

```
gunzip libc.html.gz
gzip -d libc.html
gunzip libc.html.gz User_Guide.ps.gz
```

Retrieves lib.html
Same—.gz *assumed*
Works with multiple files

You can now browse the files with their respective viewers—a browser like Netscape or Mozilla for HTML files, and **gv** for Postscript documents.

Recursive Compression (-r) Like **cp**, you can also descend a directory structure and compress all files found in subdirectories. You need the -r option, and the arguments to **gzip** must comprise at least one directory:

```
gzip -r progs
```

Compresses all files in progs

This option can be used for decompression also. To decompress all files in this directory you need to use **gunzip -r progs** or **gzip -dr progs**.

Tip: To view compressed text files, you really don't need to "gunzip" (decompress) them. Use the **gzcat** and **gzmore** (or **zcat** and **zmore**) commands if they are available on your system. In most cases, the commands run **gunzip -c**.

Note: For some years, **gzip** reigned as the most favored compression agent. Today, we have a better agent in **bzip2** (and **bunzip2**). **bzip2** is slower than **gzip** and creates .bz2 files. We are beginning to see .bz2 files on the Internet. **bzip2** options are modeled on **gzip**, so if you know **gzip** you also know **bzip2**.

5.16 tar: THE ARCHIVAL PROGRAM

For creating a disk archive that contains a group of files or an entire directory structure, we need to use **tar**. The command is taken up in some detail in Chapter 15 to back up files to tape (or floppy), but in this section we need to know how the command is used to create a disk archive. For this minimal use of **tar** we need to know these *key* options:

-c	Create an archive
-x	Extract files from archive
-t	Display files in archive
-f *arch*	Specify the archive *arch*

Only one these key options can be used at a time. We'll also learn to use **gzip** and **gunzip** to compress and decompress the archive created with **tar**.

5.16.1 Creating an Archive (-c)

To create an archive, we need to specify the name of the archive (with -f), the copy or write operation (-c) and the filenames as arguments. Additionally, we'll use the -v (verbose) option to display the progress while **tar** works. This is how we create a file archive, archive.tar, from the two uncompressed files used previously:

```
$ tar -cvf archive.tar libc.html User_Guide.ps
a libc.html 3785K
a User_Guide.ps 364K
```

-v *(verbose) displays list*
a *indicates append*

By convention, we use the .tar extension, so you'll remember to use the same **tar** command for extraction. We created an archive containing two ordinary files, but **tar** also behaves recursively to back up one or more directories. In the following example, **tar** fills the archive progs.tar with three directory structures:

```
tar -cvf progs.tar c_progs java_progs shell_scripts
```

We'll soon use the same **tar** command to extract files from this archive. But before we do that, let's see how we can compress this archive.

Using `gzip` *with* `tar` If the created archive is very big, you may like to compress it with **gzip**:

```
gzip archive.tar                                    Archived and compressed
```

This creates a "tar-gzipped" file, `archive.tar.gz`. This file can now be sent out by FTP or as an email attachment to someone. A great deal of open-source UNIX and Linux software are available as `.tar.gz` files on the Internet. To use the files in this archive, the recipient needs to have both **tar** and **gzip** at her end.

5.16.2 Extracting Files from Archive (-x)

tar uses the `-x` option to extract files from an archive. You can use it right away on a `.tar` file, the one we just used to archive three directories:

```
tar -xvf progs.tar                                  Extracts the three directories
```

But to extract files from a `.tar.gz` file (like `archive.tar.gz`), you must first use **gunzip** to decompress the archive and then run **tar**:

```
$ gunzip archive.tar.gz                             Retrieves archive.tar
$ tar -xvf archive.tar                              Extracts files
x libc.html, 3875302 bytes, 7569 tape blocks        x indicates extract
x User_Guide.ps, 372267 bytes, 728 tape blocks
```

You'll now find the two files in the current directory. Selective extraction is also possible. Just follow the above command line with one or more filenames that have to be extracted:

```
tar -xvf archive.tar User_guide.ps                  Extracts only User_guide.ps
```

This extracts a single file from the archive. If you use a pathname, it must be exactly in the same form that was used during the copying operation. This has been discussed later *(15.9.2—Tip)*.

5.16.3 Viewing the Archive (-t)

To view the contents of the archive, use the `-t` (table of contents) option. It doesn't extract files, but simply shows their attributes in a form that you'll see more often later:

```
$ tar -tvf archive.tar
-rw-r--r-- 102/10  3875302 Aug 24 19:49 2002 libc.html
-rw-r--r-- 102/10  372267 Aug 24 19:48 2002 User_Guide.ps
```

You'll understand the significance of these columns after you have learned to interpret the `ls -l` output *(6.1)*. But you can at least see the individual file size (third column) and their names (last column) in this output.

Both **tar** and **gzip** can be made to behave like *filters* (a group of programs whose input and output are quite flexible). After you have understood how filters work, you'll be able to perform both activities without creating an intermediate file.

5.17 zip AND unzip: COMPRESSING AND ARCHIVING TOGETHER

Phil Katz's popular PKZIP and PKUNZIP programs are now available as **zip** and **unzip** on UNIX and Linux systems. **zip** generally doesn't compress as much as **bzip2** but it combines the compressing function of **gzip** with the archival function of **tar**. So instead of using two commands to compress a directory structure, you can use only one—**zip**. All the letters of the alphabet are available as its options but we'll consider just a few of them.

zip requires the first argument to be the compressed filename; the remaining arguments are interpreted as files and directories to be compressed. The compression in the previous example could have been achieved with **zip** in the following way:

```
$ zip archive.zip libc.html User_Guide.ps
  adding: libc.html (deflated 80%)
  adding: User_Guide.ps (deflated 66%)
```

The unusual feature of this command is that it doesn't overwrite an existing compressed file. If `archive.zip` exists, files will either be updated or appended to the archive depending on whether they already exist in the archive.

Recursive Compression (-r) For recursive behavior, **zip** uses the `-r` option. It descends the tree structure in the same way **tar** does except that it also compresses files. You can easily compress your home directory in this way:

```
cd ; zip -r sumit_home.zip .
```

`cd` *is same as* `cd $HOME`

Using `unzip` Files are restored with the **unzip** command, which in its simplest form, uses the compressed filename as argument. **unzip** does a noninteractive restoration if it doesn't overwrite any existing files:

```
$ unzip archive.zip
Archive:  archive.zip
  inflating: libc.html
  inflating: User_Guide.ps
```