# Chapter 9

# Data Manipulation

This chapter introduces the user to the many of the commands that are available to work with and analyze data. The list is not complete – and should never be, as new commands are developed. After an introduction, the user will understand more of what commands are available to use directly, in scripts, and enhanced editors such as awk (gawk) and sed.

There are many commands noted in this chapter that might seem obsolete. That may very well be true, but it also demonstrates the heritage of the operating system. Learn to see where these commands might be useful.

**Concepts Learned in this Chapter**

- Utilities designed to manipulate data and information, thus providing a productive system


© Dennis Rice

# Table of Contents




© Dennis Rice

© Dennis Rice

There are many applications that are available to the Unix and Linux user. The design concept of each command is – Do one thing and do it well. Thus over the years, many small powerful utilities have been developed to manipulate data. This is an introduction to just a limited number of the commands available – and a few are fun, but someone else needs to explain their justification.

## 9.1 Displaying a Message to the Monitor

The Echo Utility is designed to report to the stdout (screen) what is typed in – but it has capabilities well beyond that. In general, it is one of the most useful commands available for displaying information on the screen.

Echo takes the arguments that follow the command and issues them as just a character string. If you should place another command after the echo utility, the response will be to print out the command characters, but not enact the command. For example:

$ **echo ls**

displays

***ls***

– and nothing happens.

You can place a system variable as an argument to echo, such as:

$ **echo $USER**

your response will be

***your-username***

All of the environmental variables may be displayed using the echo command.

If you need to output the literal text of a variable, then you enclose the text in either single “forward” quotes or double quotes. For example:

$ **echo ‘$USER’** or

$ **echo “$USER”**

and you will observe

***$USER***

If you need to output another command in the middle of a text string, then you need to put the command in between **BACKTICs**. This is the character that looks like a single quote to the left of the numeric 1 key. For example:

**$ echo ‘$PATH’ variable is “$PATH”** notice the different type of quote characters

So what is the value of the echo command? Being issued at the keyboard – nothing! But what if you are writing a script, and you need to instruct the user to perform an action, such as changing the CD it becomes a very powerful command. This command is particularly useful when writing scripts.


© Dennis Rice

## 9.2 Sorting a File

The **sort** utility provides a function of just what the name is – it sorts a list of items and reorganized them into a sorted list.

Sort takes as its input a file with a content list. This file is then sorted and displayed on the screen. Several options are available.

Options include:

- -o Output the sorted list to a file.
- -r Create a reverse order sorted list.
- -d Create a sorted list based on the dictionary listing, consider only blanks and alphanumeric characters.
- -u Suppresses identical lines in the original file.

Additional options exist.

The following is an example of a list of names (Original), and the output of several different sorted formats (straight, dictionary, and reverse).

| ***Names*** | ***namesaz*** | ***namesd*** | ***namesza*** |
|---|---|---|---|
| ***Original*** | ***sort*** | ***sort -d*** | ***sort -r*** |
| **Robert** | **alan** | **alan** | **william** |
| **Joe** | **bill** | **bill** | **thomas** |
| **dennis** | **dennis** | **dennis** | **steven** |
| **mike** | **Dennis** | **Dennis** | **sean** |
| **william** | **janett** | **janett** | **Robert** |
| **sean** | **joe** | **joe** | **robert** |
| **alan** | **Joe** | **Joe** | **mike** |
| **steven** | **julie** | **julie** | **kevin** |
| **kevin** | **kevin** | **kevin** | **julie** |
| **julie** | **mike** | **mike** | **Joe** |
| **janett** | **robert** | **robert** | **joe** |
| **robert** | **Robert** | **Robert** | **janett** |
| **thomas** | **sean** | **sean** | **Dennis** |
| **bill** | **steven** | **steven** | **dennis** |
| **joe** | **thomas** | **thomas** | **bill** |
| **Dennis** | **william** | **william** | **alan** |

The sort utility may also be used to append two files, with the output sorted.

## 9.3 Comparing File Contents

The **cmp (compare)** is used to compare two files, then listing the difference between the two by character position and line.

The syntax for the cmp command is:

**$ cmp file1 file2**

The output specifies the where the first byte and line error exists, rather than the differences. Only the first difference is reported.

Two options are available, –l and –s. The “l” option specifies that the difference in the files be outputted in a hex format, the “s” option specifies that


© Dennis Rice

no output be given, but that the results are returned as an error message with a 0 if there is no difference, or 1 if there is a difference. The “s” option would be used in a script when you wish to test the results, and then take action accordingly.

```
# cmp   names   namesaz
names   namesaz   differ:   byte   1,   line   1
```

## 9.4 Differences Between Two Files

The **diff (difference)** utility is used to determine differences between files or directories. Although it may produce similar results to the cmp utility, it operates in a different way. The cmp compares two files character by character, whereas the diff utility compares line by line. The cmp utility is most useful for comparing binary files rather than text files.

The syntax for diff is:

$ **diff file1 file2**

There are three options for this utility:

-i Ignore differences in case
-q Provides a summary of information, only if they do differ.
-b Ignore changes in whitespace.

```
# diff   names   namesaz
1,2c1,2
< Robert
< Joe
---
> alan
> bill
3a4,9
> Dennis
> janett
> joe
> Joe
> julie
> kevin
5c11,12
< william
---
> robert
> Robert
7d13
< alan
9,12d14
< kevin
< julie
< janett
```


© Dennis Rice

***< robert***
***14,16c16***
***< bill***
***< joe***
***< Dennis***
***---***
***> william***

To fully utilize the full capability of this command, study the **man** page to learn all of the options.

## 9.5 Counting Words, Lines, and Characters

The **wc (word count)** utility provides a count of characters, lines, and words of a file. All three may be printed out, or one at a time.

Syntax for the wc command is:

$ **wc filename**

Options for only one output are:

-l Prints the number of lines in the file.
-c Prints the number of characters in the file.
-w Prints the number of words in the file.

If only the wc command is used without options, then the response will be of the format:

**wc names**
***16 16 97 names***

In this example, the file names has 16 lines with 16 words and 97 characters.

## 9.6 Extracting Data from a File

The utility **cut** allows us to extract a tab delimited character range or fields from a file. Using this we can output selected parts of lines from a file to the monitor.

The syntax is:

$ **cut –{option} filename**

where option include:

**b** byte range -b10-20
**c** character range -c25-50
**d** specify delimiter character -d:
(must follow -f option)
**f** tab delimited field range -f5-8
(**tab** is the default delimiting character)
**s** ignore lines that do not contain the tab character to delimit the fields


© Dennis Rice

The range value may be a single value, from a value to the end of the line, or between two values. Additionally, multiple ranges may be specified, when comma delimited. For example:

| | |
|---|---|
| **-b10-20** | outputs the character numbers 10 through 20 |
| **-b10,20** | outputs the character numbers 10 and 20 |
| **-b10-** | outputs the character number 10 to the end of the line |
| **-b10-20,30-40** | outputs two byte ranges, note no space between ranges |
| **-c25-50,60-80** | outputs two character ranges, note no spaces between ranges |

As an example, we can use the /etc/password file extract data from. Only a portion of the list is shown.

| **-c1-10** | **-f1-2 -d:** | **-f1,4-5 -d:** |
|---|---|---|
| ***xfer:x:500*** | ***xfer:x*** | ***xfer:500:*** |
| ***dennis:x:5*** | ***dennis:x*** | ***dennis:501:*** |
| ***kenny:x:50*** | ***kenny:x*** | ***kenny:502:*** |
| ***keller:x:5*** | ***keller:x*** | ***keller:503:*** |
| ***russ:x:504*** | ***russ:x*** | ***russ:504:*** |
| ***hlul:x:505*** | ***hlul:x*** | ***hlul:505:*** |
| ***wcc:x:506:*** | ***wcc:x*** | ***wcc:506:*** |
| ***linuxfer:x*** | ***linuxfer:*** | ***linuxfer:507:*** |

## 9.7 Combining Files Together

A user may combine text together, line by line, by using the command **paste**. Probably the best example would be to combine common records together from a database for a special application.

The command for combining files is:

**$ paste file1 file2**

For example, say you have two files, one with a name and the second with the residence, such as:

Name File:
1234 Doe John
2345 Buyer Mike

Residence File:
1234 Irving
2345 Arlington

Now issuing the command:

**$ paste Name-File Residence-File**
***1234 Doe John 1234 Irving***
***2345 Buyer Mike 2345 Arlington***


© Dennis Rice

## 9.8 Displaying and Setting the System Date

The **DATE** utility is used to display the current day, date, and time or for correcting the system date and time.

The utility provides many options for displaying portions of what is needed, so that you may use the information in a script.

### 9.8.1 Displaying a Date

To display the date, issue the command:

$ **date**
***Tue Jan 10 10:18:47 CST 2006***

lists the present day, time, and date as computed from the PC clock. Note that this may not be accurate!

Issuing the command:

$ **date +%*y***
**06**

To display just the last two digits of the year. Note that you must type in the "**+**" character. To display the year in century format, use a capital Y.

**$ date +%Y**
***2006***

Issuing the command:

$ **date +%m%Y**
***012006***

displays the present month and year only. Note that they run together and there is no space. To insert a space between the month and year, a space must be inserted with quotes.

**$ date +%m" "%Y**
***01 2006***

To display the date in a written out format, one might use:

**$ date +%A ", "%B " " %d " " %G " " %r**

Note that there must be no spaces between the options, that spaces between options must be in quotations. There are many additional options, check them out in the **man** page.

### 9.8.2 Setting the Time and Date

To set the time and date of the computer, issue the following command:

**# date MMDDhhmm[cc]YY[.ss]**

where

MM is the month, 01 – 12
DD is the day of the month, 01 – 31
hh is the hour of the day in military format, 01 – 24
mm is the minute of the hour, 01 – 59


© Dennis Rice

[cc] is the century, optional value is either 19 or 20, must specify for a different century
YY is the year, 00 – 99
[.ss] is the seconds of the minute, optional if you know how to account for propagation time

As an example, if we wish to set the date to January 12, 2006, 3:15 PM, one would issue the command:

**# date 01120615152006**

Note that only the root administrator may modify the system date.

## 9.9 Displaying a Calendar

The **CAL** utility is often used to evaluate project due dates – or for figuring out special days.

The utility provides three calendar formats – year, month and today's date.

Issuing the command:

**cal**

lists the full year calendar for the present year.

Issuing the command:

**cal *year***

displays a full annual calendar for the specified year. If the year is specified with only the last two digits, then you may very well receive an error for the year. To make sure of what you are doing, specify all four digits for the year.

Issuing the command:

**cal *month year***

where month is a numeric value, displays the specified month and year only.

Just for the curious, you might wish to display the calendar for the year 1752, and explain what happened in September.

## 9.10 Comparing Left and Right Files

After collecting and sorting data, we may need to compare two different files line by line to compare for differences. This can be accomplished by using the command:

**comm (option) File-Left File-Right**


© Dennis Rice

This compares two sorted files on a line by line basis. Options include:

| | |
|---|---|
| -1 | Suppress lines that are unique to the File-Left |
| -2 | Suppress lines that are unique to the File-Right |
| -3 | Suppress lines that appear in both files |
| --help | Display help file |
| --version | Display the version information |

Lets demonstrate with two simple files:

**$ cat > lf** and **$ cat > rf**

| lf | rf |
|---|---|
| **a** | **a** |
| **b** | **b** |
| **c** | **d** |
| **e** | **f** |
| **f** | **g** |
| **g** | **n** |
| **m** | |

**$ comm lf rf**

| Left Only | Right Only | Both |
|---|---|---|
| | | ***a*** |
| | | ***b*** |
| ***c*** | | |
| | ***d*** | |
| ***e*** | | |
| | | ***f*** |
| | | ***g*** |
| ***m*** | | |
| | ***n*** | |

**$ comm –1 lf rf**

| Right | Both | If line is in Left Only, line suppressed |
|---|---|---|
| | ***a*** | |
| | ***b*** | |
| ***d*** | | |
| | ***f*** | |
| | ***g*** | |
| ***n*** | | |

**$ comm –2 lf rf**

| Left | Both | If line is in Right Only, line suppressed |
|---|---|---|
| | ***a*** | |
| | ***b*** | |
| ***c*** | | |
| ***e*** | | |
| | ***f*** | |
| | ***g*** | |
| ***m*** | | |


© Dennis Rice

**$ comm -3 lf rf**

| Left | Right | Suppress lines that are common to both files |
|---|---|---|
| ***c*** | | |
| | ***d*** | |
| ***e*** | | |
| ***m*** | | |
| | ***n*** | |

## 9.11 User Dictionary Utilities

Unix and Linux include a user dictionary that may be accessed and utilized to check your documents.

The base dictionary is located in the /usr/share/dict/words file. There are three utilities that we may utilize when verifying the spelling of a file. These are:

1. look
2. spell
3. ispell

### 9.11.1 look

The first utility provides us the ability to look up words in our dictionary file. The syntax is:

$ **look string [/path/file]**

Where we wish to look up words in our dictionary that start with the specified string. By default the /usr/share/dict/words file is used, but an alternative file may be specified. The output is a list of words that match what you are looking for.

**# look command**

| | | |
|---|---|---|
| ***command*** | ***commander*** | ***commandments*** |
| ***commandable*** | ***commanderies*** | ***commando*** |
| ***commandant*** | ***commanders*** | ***commandoes*** |
| ***commandants*** | ***commandership*** | ***commandoman*** |
| ***commandatory*** | ***commandery*** | ***commandos*** |
| ***commanded*** | ***commanding*** | ***commandress*** |
| ***commandedness*** | ***commandingly*** | ***commandrie*** |
| ***commandeer*** | ***commandingness*** | ***commandries*** |
| ***commandeered*** | ***commandite*** | ***commandry*** |
| ***commandeering*** | ***commandless*** | ***commands*** |
| ***commandeers*** | ***commandment*** | |

### 9.11.2 spell

The spell utility provides a check of a file and provides a list of misspelled words. The syntax is:

**$ spell file**

***$* cat testfile**
***Now si the tim for all good meen.***


© Dennis Rice

***$ spell testfile***
***meen***
***si***
***tim***

### 9.11.3 ispell

The ispell utility add to the spell utility by prompting the user with a list of possible words so that update / corrections may be made immediately. The syntax is:

$ **ispell file**

***$* ispell testfile**
***Now si tim for all good meen.***

| | |
|---|---|
| ***1) Si*** | ***6) sir*** |
| ***2) SO*** | ***7) so*** |
| ***3) Sir*** | ***8) is*** |
| ***4) Sui*** | ***9) S*** |
| ***5) sci*** | ***0) s*** |
| ***i) Ignore*** | ***I) Ignore all*** |
| ***r) Replace*** | ***R) Replace all*** |
| ***a) Add*** | ***l) Add Lower*** |
| ***b) Abort*** | ***x) Exit*** |

**?**

At the "?", type is the desired option. The selected option will then replace the improper characters.

## 9.12 Multiple Outputs from a Command

Two utilities are available that allow the output of a command to be branched in two different directions at the same time. The intent is slightly different, but both accomplish the same effect. The goal is to take the output from a command and direct it to a file and to either the desired file and the screen.

### 9.12.1 tee Utility

It is often convenient to have the output of another command outputted to both the screen and to a file, for later observation. This is accomplished using the tee command.

The syntax of the tee command is:

$ **tee [option] filename**

The options are:

-a Append the output to an existing file
-i Ignore interrupts


© Dennis Rice

As an example, if we wish to record the output of the info command to include in another write-up, while observing it on our monitor, we could use the following:

**$ info sudo | tee sudofile**

You will now have a new file created - sudo in the present directory.

### 9.12.2 Logging a Session's Commands

While issuing commands, it is often convenient to log, or record, your actions and responses to a file. At a later time, this file may be printed out for review.

The format of the command is:

**$ script [option] filename**

The options include:

**-a** Append the session to an existing file.
**-f** Flush the output from the buffer to the file after each disk writing.

To terminate the script session, issue a CTRL-D (^D) at the bash or sh shell prompt. Commands that manipulate the screen (such as a text editor like vi or pico) may produce garbage in the script file.

**$ script scriptfile.txt**
***Script started, file is scriptfile.txt***
**. . .**
**$ Whatever commands you issue.**
**. . .**
**$ ^D**
***Script is done, file is scriptfile***

## 9.13 Batched Commands

It is often convenient to create a file with a list of commands that one would issue multiple times – thus not having to retype all of the commands over and over. It is essentially the same as the batch command file used in Microsoft, although it has several advantages over that used by Microsoft. The user may issue the command by itself, or could create a periodic command using **at**. The set of commands may be issued when the processor load drops below an average of 0.8, or at the designated time. The syntax of the command is:

**$ batch -f filename [time]**

For example, if we have the following file (batchfile) of commands:

```
cut -d: -f1,3-5 passwd > passb1
cut -d: -f7 passwd > passb2
paste passb1 passb2 > passb3
```

Now every time this set of commands are issued, a new file is created that contains only the username, userid, groupid, and the home directory. This file may be executed in the same way that the "**at**" command was, that is at a specified time, such as:


© Dennis Rice

**$ batch batchfile 1404**
***job 1 at 2006-01-12 14:04***

An alternative to issuing the command at the specified time, one can also have the batch command file issued immediately. This is accomplished by issuing the command:

**$ source batchfile**

## 9.14 Changing GroupID

The **newgrp** command changes the group identification of its caller. Effectively, a user by issuing the newgrp command takes on the groupid of the specified group, assuming they are a member of that group. The command format is:

**$ newgrp groupname**

To change back to the user's personal gropupid, the user needs to issue the command without specifying a groupname.

## 9.15 File Verification

When we download a file off of the Internet, it is an excellent idea to verify that the file has been received correctly – that no errors have been incurred. There are two alternatives to file verification, cksum and md5sum.

When you find a file that you wish to download, you will commonly observe that it also includes what is called a checksum value. What you do is to document this value, download the file, run a checksum program against the downloaded file, and finally verify that the computed checksum value is the same as the one you previously documented.

The application etherape-0.8.2-3.i386.rpm has been downloaded from rpmfind.net as an example for demonstration. From the html page we would find the MD5 sum of ec9f4404a1c3c2bf67baba22613f1801

### 9.15.1 cksum

To compute the checksum for a file you issue the command:

**$ cksum [option] filename**

cksum computes the Cyclic Redundancy Check (CRC) using the POSIX.2 algorithms standard. It is not compatible with the BSD or System V "sum" algorithm, but is more rebust.

The only two options include:

--help
--version


© Dennis Rice

Running cksum on etherape gives the following:

**$ cksum etherape-0.8.2-3.i386.rpm**

***3834393926 127489 etherape-0.8.2-3.i386.rpm***

This should be expected, because the download specified an MD5 Checksum.

### 9.15.2 md5sum

To compute or check the md5sum of a file, you issue the command:

**$ md5sum [option] filename**

md5sum computes the 128 bit checksum based on the algorithm specified in RFC 1321. It computes the value from the specified file or directly from the standard input (keyboard).

Options include:

| | |
|---|---|
| -b | Read files in binary mode |
| -c | Check MD5 sums against a specified list |
| -t | Read files in text mode (default) |
| -w | Warn about improperly formatted checksum |
| --status | Output status code |
| --help | |
| --version | |

Running the md5sum against the etherape application gives us:

**$ md5sum –c etherape-0.8.2-3.i386.rpm**

***md5sum: etherape-0.8.2-3.i386.rpm: no properly formatted MD5 checksum lines found***

**$ md5sum –b etherape-0.8.2-3.i386.rpm**

***a4418aa36142f7fc6a88c93d2e932bfc *etherape-0.8.2-3.i386.rpm***

**$ md5sum etherape-0.8.2-3.i386.rpm**

***a4418aa36142f7fc6a88c93d2e932bfc etherape-0.8.2-3.i386.rpm***

### 9.15.3 sha1sum

The third option for checking the file's validity is **sha1sum**. This is a relatively new verification option. MD5sum has been found to have a vulnerability and may be updated in the future.

To generate an sha1sum is not a direct as it was for the other two options. Calculating the sum utilizes the **openssl** utility. **Openssl** is a very rich cryptography utility that is capable of generating many different encryption hashes, including md5, des, des3, dsa, and many others. It is also capable of acting as a Certificate Authority, capable of issuing certificates to remote systems for authentication purposes. To calculate the sha1sum for a file, issue the following command:

**$ openssl sha1 filename**

***SHA1(filename) = f91842ae84074d82a738d7503c7495b193792756a***

The sum generated is 40 characters long, in hexadecimal format. We will see much more of the use of openssl when security is an issue.


© Dennis Rice

## 9.16 Column Manipulation

When working with text, columns may be either be created or removed, depending upon the users requirements.

### 9.16.1 Creating Columns from Data

The **Column** utility allows one to convert a single column list to multiple columns. One option exist to allow control of the utility. The syntax of the utility is:

$ **column [-x] filename**

The output is to the standard output – normally the monitor.

If a filename is not specified, then input is taken directly from the keyboard. After the last item is typed in, on a new line key in a CTRL-D (^D). The list will then be re-displayed in column format.

As an example, if we issue the command (taking input directly from the keyboard):

```
$ column
1
2
3
4
5
^D
```

results in:

***1 2 3 4 5***

Additional options exist, which the reader is referred to the man page. (Note – the "-c" option appears to not work as specified.

### 9.16.2 Removing Columns from Data

Using the **colrm**, one can remove columns from data that has been previously formatted in a columnar format.

The format of the command is:

**$ colrm [startcol] [endcol]**

If the command is issued with just one parameter, then columns starting with the first value will be removed. If two parameters are specified, then the columns starting with the first value and ending with the last value will be removed.

Each character position is considered a column, as demonstrated in the following examples.

```
$ colrm 4
1234567890
123
$ colrm 4 6
```


© Dennis Rice

**1 2 3 4 5 6 7 8 9 0**
***1 2 3 7 8 9 0***

One may also take input from a file by using the input redirector. For example, create a file with the contents:

**$ cat > colfile**
**This is the time for all to learn Linux.**
**^D**
**$ colrm 9 17 < colfile**
***This is for all to learn Linux.***

## 9.17 Controlling a ZIP Drive

In like manner to the mtools, one can also issue special commands to a zip disk.

The command format is:

**$ mzip [option]**

Options include:

| | |
|---|---|
| **-e** | Eject the disk. |
| **-f** | Force an eject even if the disk is mounted (must be issued along with the **e** option. |
| **-r** | Write protect the disk. |
| **-w** | Remove the write-protection. |
| **-x** | Password protect the zip disk. |
| **-u** | Temporarily unprotect the disk until it is ejected. The disk is writable and reverts back to its protected status when ejected. |
| **-q** | Queries the zip disk status. |

In order to remove a password protected zip disk, set it to a password-less mode (-r or –w), at which time you will then be queried for the password. If you forget the password, all data will be lost as to reuse the disk, you will have to perform a low level formatting. Note that this command may not be supported on more modern units.

## 9.18 Converting Tabs to Spaces

When reading a document we may find that it contains the TAB character, but we would prefer to have white spaces instead. This is accomplished by issuing the command:

**$ expand [options] filename**

Options include:

| | |
|---|---|
| **-i** | Do not convert TABs after a non-whitespace. |
| **-t** | Set tabs specified number of characters apart rather than the default 8. |


© Dennis Rice

**expand** maintains the same spacing as if a tab still exists, thus maintaining the spacing.

Output of **expand** is normally to the standard output (monitor). If you want to create a new file, you need to use the director (>) to create a new file.

An example of the usage is the file expandfile:

**1 2 3 abc def**
**1234567890 abc def**

where the spacing between characters is a TAB.

The output of the above file would be:

**$ expand filename**
***1 2 3 abc def***
***1234567890 abc def***

There is no obvious difference in the text spacing, but the tab has been replaced with the appropriate number of spaces to maintain proper alignment.

## 9.19 Converting a Text File to Postscript

In the process of creating text files, such as documentation, it is convenient to convert the file to Postscript format. Normal output may be either to a file or directly to a printer. There are numerous options, of which only a few will be covered in this review.

The syntax of the command is:

**$ enscript [option] input-file**

Some options includes:

| | |
|---|---|
| **-#** | Print the specified number of copies. |
| **-1, -2** | Specify the number of columns to be printed. |
| **-a pages** | Print the specified pages. Additional options exist for this. |
| **-b *header*** | Add a page header string as specified by ***header***. |
| **-c** | Truncate lines, by default, lines are wrapped. |
| **-d *name*** | Spool printer to printer ***name***. |
| **-v** | Verbose mode – tell what enscript is doing. |
| **-W *lang*** | Generate output in specified language.<br>Postscript (default)<br>html – generate html code<br>overstrike – generate overstrikes<br>rtf – generate Rich Text Format |
| **-toc** | Print a Table of Contents at the end of the printjob. |
| **-p *filename*** | Output file to file ***filename***. |

## 9.20 Displaying the Last N Issued Commands

An alternative to the history command is **fc**. This will list out a limited set of history commands, depending on the specified option. The command syntax is:

**$ fc (option) (first) (last)**


© Dennis Rice

**fc** issued by itself will re-issue the last entry. If the entry was not a command, then the vi line editor will be invoked. Recall that the way out of vi is {ESC:q}

Options include:

**-l** Display only the last 17 commands by default
**-l –10** Displays the last ten command (value is variable)
**-lr** Displays the last 17 commands in reverse order
**-ln** Suppress command line numbers

An example of its usage is:

```
# fc -l 101 110
101   cd /qos
102   ssh gateway
103   cd /etc/sysconfig/network-scripts/
104   ls
105   cat ifcfg-eth0
106   cd /etc
107   cat resolv.conf
108   clear
109   cat resolv.conf
110   clear
```

## 9.21 Group Password

The **/etc/group** file may be administered by issuing the command:

**gpasswd [option] group-name**

Additional options are available that allow users to be added to a group, and a group administrator may be created. This would relieve the system administrator from having to provide services to a specific group.

Options include:

| | |
|---|---|
| **gpasswd group** | Prompts the administrator for a new group password. |
| **gpasswd -a user group** | Add a user to the specified group. |
| **gpasswd -d user group** | Delete a user from the specified group. |
| **gpasswd -r group** | Removes the password from the group. |
| **gpasswd -A user group** | Specifies the user as an administrator of the group. |
| **gpasswd -M user group** | Specifies the user as a member of the group and deletes other members except for the administrators. |
| **gpasswd -R group** | Disables access to the group ID through the **newgrp** command. |

An example of using the gpasswd command is:

```
$ gpasswd test          Add a password to the group test
Enter password:
Re-Enter password:
```


© Dennis Rice

**$ gpasswd -A dennis test** Add user dennis as an administrator
**$ gpasswd -a brian test** Add user brian
**$ gpasswd -a russ test** Add user russ
**$ cat group | grep test** Display the group test
***test:x:507:brian,russ***
**$ cat gshadow | grep test** Display the group shadow for group test
***test:Lj5U3UBverz7.:dennis:brian,russ***

## 9.22 Learning the Login Name

Within a script it may be necessary to learn who the application is running under. This can be easily achieved by issuing the command:

$ **logname**
***dennis***

## 9.23 Listing Directory Contents with DIR

The command **dir** is a holdover from the DOS Operating System. It is completely equivalent to the **ls** command.

**$ dir**

## 9.24 Logging in as Another User

The command **login** allows a user to change from one login name to another. This is similar to Switch User (**su**), but is a permanent login change. The command format is:

$ **login**

The effective action of the command is to log out as the present user and issue a new login prompt. Changing from a normal user to the administer is not permitted.

## 9.25 Merging 3 Files

Three files can be merged together by using the **merge** command. This can be useful when you have your original file and two other files that have modifications to the original.

Merge incorporates all of the changes that were made to file-3 that originated from file-2 into file-1. Normal operation is for file-2 to be the original, file-3 has modifications, and the output is written to file-1. If file-1 originally also has changes from file-2, then the changes from both file-1 and file-3 will be written to written to file-1.

A simple example is required to demonstrate what happens. Because of the order of the files being combined, we will set up the following files:


© Dennis Rice

File1:
**Now is the time**
**For all good men**

File2:
**Now is the time**

File 3:
**For all good men to come**
**to the aid of their party.**

Now issuing the command:
**$ merge File2 File1 File3**

Results in:
***merge: warning: conflicts during merge***
The new File2 has the contents:
***<<<<<<< File2***
***Now is the time***
***=======***
***for all good men to come***
***to the aid of their party.***
***>>>>>>> File3***

When files 1 and 3 have changes in common between the lines, it is common to output a warning to the monitor and to include the changes into file 1 with the warning message. The above is an example shows the warning. The user must manually edit the file to remove the warning message.

## 9.26 Detecting Mouse Clicks

When in a text mode, either in a full CLI or Xterminal, one can detect mouse clicks and wheel scrolling, but not movement. The output is typically to the stdout (monitor), but may be redirected to other applications. Additional research on the part of the user is required to determine if this command might be useful to their application.

When the command **mev** is issued at a run level 3, various characters will be outputted to the monitor indicating clicking of the mouse buttons or changes to the scroll wheel. If in an Xterm session, then the command **rmev** must be used.

The command syntax is:
**$ mev**

There is no special options, you will see the a text line, repeated every time you move the mouse. The line will have the format something like:
***mouse: event 0x01, at 74,14 (delta -1, 0), buttons 0, modifiers 0x00***

This tells you:
Coordinates (74,14) specifies the mouse ball location


© Dennis Rice

(delta -1, 0) specifies the change from the previous location
Button pressed:

| | |
|---|---|
| Left button: | 4 |
| Right button: | 1 |
| Middle button: | 2 |
| Both buttons: | 5 |
| Scroll roller: | 0 |

This is repeated every time the mouse ball is moved or a button is clicked. The action is terminated with the clicking of Ctrl - C.

## 9.27 Modifying a Command's Priority

When issuing a set of commands that are conflicting in time resources, it may be necessary to alter the priority of one or more commands, allowing one to operate with more processor cycles than another. Any user may reduce the priority of an application, but only the administrator may increase its priority.

The command syntax to modify the priority of a command is:

**$ nice (option) command [arguments]**

The one functional option is **–n**, where n is a value between –20 (lowest) and 19 (highest).

An instance where this would be helpful might be when compiling programs, and you want to perform other activities, and would set the compiling process to a lower priority.

## 9.28 Numbering the Lines of a File

The command **nl** numbers each line of a file. The syntax of the command is:

**$ nl [option] file**

Issuing the command without an option will display the file contents (like cat), but with line numbers prefixed to each line.

Options include:

**-ba** Number all lines
**-bt** Number only non-empty lines (default)
**-bn** Number no lines

Additional options are available that need to be reviewed
As an example, say we have the file '**nlfile**':

**This is the time for all to learn Linux.**

**One needs to study hard to learn the commands.**

Our output would then be:


© Dennis Rice

```
$ nl   nlfile
1       This is the time for all to learn Linux.

2       One needs to study hard to learn the commands.
```

Then issuing the command:

```
$ nl   -ba   nlfile
1       This is the time for all to learn Linux.
2
3       One needs to study hard to learn the commands.
```

## 9.29 Format a File for Printing

It is often convenient to format a file for setup to the printer. The **pr** command allows the user to perform this task. Using **pr** the user may format text in columnar mode, printing across rather than vertical, double space text and many options. The syntax of the command is:

**$ pr [option] file**

Options include:

| | |
|---|---|
| **-COLUMN** | Output the specified number of columns, balancing the number of lines on each page |
| **-a** | Print columns across rather than down, used with the –COLUMN option |
| **-c** | Show control characters |
| **-d** | Double space text |
| **-F, -f** | Use form feed to separate pages |
| **-h HEADER** | Create a header with the text of HEADER |
| **-l n** | Set page length of output to n lines |
| **-o n** | Set margin width to n spaces |
| **-W n** | Set page width to n characters |

Many additional commands options exist that the user should investigate.

## 9.30 Precision Calculator

**dc** is a high precision desktop calculator that utilizes the "reverse-polish-notation (RPN)"[1] technique to perform its calculation – that is, there is no ENTER key. This process is identical to how a computer solves numbers, and what is used on calculators manufactured by Hewlet-Packard. What is hard to get use to is that there is no equal sign used during the calculations. Calculations to be performed may be entered directly from the keyboard or from a file. The command format is:

$ **dc [filename]**

[1] RPN is named after a mathematician of polish decent - whose name no one can pronounce. Hence it was called Reverse Polish Notation.


© Dennis Rice

The concept of how the reverse-polish technique works is best illustrated by example. When **dc** is used, it sets up an area in memory called a **stack**. When the first value is keyed into the system, it is stored, or pushed, into the first stack position, stack1. When the second value is keyed in, the first value is "pushed" down to into the stack position 2, stack2, and the new value is pushed into stack1. If one wishes to then add the two values, the "+" key is then keyed in, and the values are added together and the resultant is pushed into stack1.

Many options are available, and only a few of the basic ones are covered here, refer to the man pages for additional options.

- **p** Prints the value of stack1.
- **f** Prints the values of all stacks.
- **N** Prints the value of stack1, then deletes the value and pulls the values up by one unit from lower stacks.
- **+** Adds the values of stack1 and stack2, replacing stack1 with the resultant.
- **-** Subtracts the value of stack1 from stack2, replacing stack1 with the resultant.
- ***** Multiplies the values of stack1 and stack2, replacing stack1 with the resultant.
- **/** Divides the value of stack1 by stack2, replacing stack 1 with the integer quotient. The remainder is not displayed.
- **%** Divides the value of stack1 by stack2, replacing stack 1 with the integer remainder.
- **~** Divides the value in stack1 by the value in stack2. The quotient is pushed into stack1, then the remainder is pushed into stack1, which pushes the quotient into stack2.

The following is a simple example of the process of using the dc calculator with input from the keyboard.

```
$ dc
10
20
+
p
30                  Add 10 and 20, print the result

50 46 – p
4                   Subtract 46 from 50, print the results

18 6 * p
108                 Multiply 18 by 6, print the results

605 25 / p
24                  Divide 605 by 25, print the quotient with no remainder

605 25 % p
5                   Divide 605 by 25, print only the remainder
```


© Dennis Rice

```
605 25 ~ f
5
24
5
24
108
4
30
$
```

Divide 605 by 25, print the value of all stacks, the first two contain the quotient and remainder of the division

## 9.31 Prime Factors of a Number

Every once in a while, we need to know the prime factors of a number. This is just what you needed when you were back in the 5th grade – oh well, can't win them all. To output the primes, issue the command:

**$ factor number**

The only options for this command are:

**$ factor --help** Specifies help for command.
**$ factor --version** Specifies the version of the command.

As an example of the command:

```
$ factor 312
312: 2 2 2 3 13
```

## 9.32 Reversing Text Output

Every once in a while, there arises a need to do something weird with the text. This is one of them, which I am sure there was a good need, but I am not sure for what. I have been told that it has been used for reversing one's password, thus making it harder to decrypt.

If you wish to reverse the text – this is a good command. Issuing the command:

**$ rev filename**

This causes the text to be replicated in reverse order on the monitor. If the text is to be saved, then the output must be directed (>) to a new file.

An example might be:

File:

```
Now is the time
to come to the
aid of our
America. It is a
time of need.
```

The output would be:

```
$ rev file
emit eht si woN
```


© Dennis Rice

***eht ot emoc ot***
***ruo fo dia***
***a si tl .aciremA***
***.deen fo emit***

This command was originally designed to reverse a user's password before it was submitted. Beyond that, you can now Have Fun!

## 9.33 Displaying a Sequence of Numbers

The command **seq** displays a sequence of numbers. The format is one of three forms:

**$ seq LAST**
**$ seq FIRST LAST**
**$ seq FIRST INCREMENT LAST**

Although additional features are available (refer to the man pages), the typical output is as follows:

```
$ seq 5
1
2
3
4
5

$ seq 5 10
5
6
7
8
9
10

$ seq 5 5 30
5
10
15
20
25
30
$
```

## 9.34 Serial Port Statistics

When developing a program or script, it may be beneficial to monitor the signal / pin status of a serial port. A simple utility to do this is:

**$ statserial**


© Dennis Rice

The status of each line is updated once each second by default, but may be set to not update if desired. Utilization of this utility would be for testing the serial interconnection to other equipment, such as a modem or router.

Issuing the statserial command results in the following:

**$ statserial**

***Device: /dev/ttyS0***

| **Signal Name** | **Pin (25)** | **Pin (9)** | **Direction (computer)** | **Status** | **Full Name** |
|---|---|---|---|---|---|
| ***FG*** | ***1*** | ***-*** | ***-*** | ***-*** | ***Frame Ground*** |
| ***TxD*** | ***2*** | ***3*** | ***out*** | ***-*** | ***Transmit Data*** |
| ***RxD*** | ***3*** | ***2*** | ***in*** | ***-*** | ***Receive Data*** |
| ***RTS*** | ***4*** | ***7*** | ***out*** | ***1*** | ***Request To Send*** |
| ***CTS*** | ***5*** | ***8*** | ***in*** | ***1*** | ***Clear To Send*** |
| ***DSR*** | ***6*** | ***6*** | ***in*** | ***1*** | ***Data Set Ready*** |
| ***GND*** | ***7*** | ***5*** | ***-*** | ***-*** | ***Signal Ground*** |
| ***DCD*** | ***8*** | ***1*** | ***in*** | ***1*** | ***Data Carrier Detect*** |
| ***DTR*** | ***20*** | ***4*** | ***out*** | ***1*** | ***Data Terminal Ready*** |
| ***RI*** | ***22*** | ***9*** | ***in*** | ***1*** | ***Ring Indicator*** |

Over time, presuming that the serial port is operational, the Status value will change as the data flows across the network. It only samples the data about once a second, so the actual may not show actual real time data transfer.

## 9.35 Splitting a File

There are times when a file may be too long, requiring them to be split into multiple parts. This may be required for transmission purposes, storage, or possibly even security.

There are two alternatives to splitting a file, **csplit** and **split**.

### 9.35.1 Splitting a File by Pattern

The format of the command to split a file by a specified pattern is:

**$ csplit [options] File Pattern**

The output of the original file are differentiated by the Pattern.

Where a file is split is specified by the Pattern, the first file will start with line 1 and continue up to the line number specified by the pattern number. Thus a file that is 10 lines long, with a pattern of 5, will be split as two files, the first being 4 lines and the second starting with line 5 through 10. If one wishes to divide the file into three parts, with a pattern of 4 and 8, the the first file will contain 3 lines (1-3), the second 4 lines (4-7), and the last 3 lines (8-10). Naturally, the each pattern number must be greater than the previous one.

The default output of the utility is to a set of files that are of the form:

**xxYY**


© Dennis Rice

where 'xx' is the file name header and 'YY' is modified to represent the file sequence. The first value of YY is '00'.

Options include:

| | |
|---|---|
| **-f PREFIX** | Creates the output file with the specified prefix name. |
| **-b SUFFIX** | Creates the output file with the specified suffix name. |
| **-n #** | Creates the output file with the specified number of digits, the default is 2. |
| **-k** | Keep any files created in the event of an error. Files are normally deleted if there is an error. |
| **-z** | Suppress creation of files that are zero length. |

Lets start by creating a file that we wish to split:

**Now is the time**
**to come to the**
**aid of our**
**America. It is a**
**time of need.**

Issuing the command:

**$ csplit file '3'**

creates:
xx00 file:

***Now is the time***
***to come to the***

xx01 file:

***aid of our***
***America. It is a***
***time of need.***

Issuing the command:

**$ csplit file '2' '4'**

creates:
xx00 file:

***Now is the time***

xx01 file:

***to come to the***
***aid of our***

xx02 file:

***America. It is a***
***time of need.***


© Dennis Rice

### 9.35.2 Spitting a File by Length

A file may be split into fixed length size files. The output files are designated as **xyy**, where by default, the base filename is 'x' and the yy extension is the characters of aa, ab, ac …, changing the character as necessary. The default size of the output file is 1000 lines in each file, with remainder in the last file. The format of the command is:

**$ split [optons] Filename [prefix]**

You may modify the output filename by specifying the prefix. Options include:

| | |
|---|---|
| **'-b BYTES'** | Size of output file in Bytes. |
| **'-C BYTES'** | Size of output file in a number of complete Lines less than the specified number of Bytes |
| **'-l LINES'** | Size of output file number of lines. |

Lets start with the file:

```
$ cat file
Now is the time
to come to the
aid of our
America. It is a
time of need.
```

To illustrate with examples. To understand what is happening, you need to remember to count spaces and new-line characters (MS uses Linefeed / Carriage Return). To illustrate, a substitution will be shown for the space (^) and newline ($):

```
$ split -b 15 file
$ cat xaa
Now^is^the^time^^$

$ cat xab
to^come^to^the$

$ cat xac
aid^of^our$
Amer

$ cat xad
ica.^^It^is^a^$

$ cat xae
time^of^need.$
```

Notice that the prompt is typically appended to the end of a line, this is due to the fact that the character at the end of the number of specified Bytes is not a Line Feed.


© Dennis Rice

```
$ split   -C   15   file
$ cat   xaa
Now is the time  --

$ cat   xab

$ cat   xac
to come to the

$ cat   xad
aid of our

$ cat   xae
America.  It is

$ cat   xaf
 a

$ cat   xag
time of need.
```

And for our last example:

```
$ split   –l   3   file
$ cat   xaa
Now is the time
to come to the
aid of our

$ cat   xab
America.  It is a
time of need.
```

## 9.36 Terminal Connectivity

The **tty** command displays the terminal that the stdin is connected to. When connected to the default terminal from boot, the response would be:

```
$ tty
/dev/tty1
```

If you should switch to an alternate terminal (ALT-Fx), then the output will reflect which terminal is presently in use (ALT-F2 outputs /dev/tty2).

## 9.37 Testing a Condition

When creating scripts, it is often necessary to test the status of variable. This is one of the most important tasks required of a network administrator when administering a network. A variable may be a condition, or the existence of a file


© Dennis Rice

or various other conditions. It is primarily intended to test file types and to compare values.

The syntax of the command is:

**$ test expression** or
**$ test option**

An expression may take on many options. All of the following output a TRUE status.

| Condition | If Condition Is |
|---|---|
| (Expression) | True |
| ! (Expression) | False |
| Expr1 -a Expr2 | True and True |
| Expr1 -o Expr2 | True or True |
| -n STRING | Length of STRING is not zero |
| -z STRING | Length of STRING is zero |
| STRING1 = STRING2 | STRINGs are equal |
| STRING1 != STRING2 | STRINGs are not equal |
| INTEGR1 -eq INTEGR2 | Integers are equal |
| INTEGR1 -ge INTEGR2 | Integer1 is greater than or equal to Integer2 |
| INTEGR1 -gt INTEGR2 | Integer1 is greater than Integer2 |
| INTEGR1 -le INTEGR2 | Integer1 is less than or equal to Integer2 |
| INTEGR1 -lt INTEGR2 | Integer1 is less than Integer2 |
| INTEGR1 -ne INTEGR2 | Integer is not equal to Integer2 |
| FILE1 -ef FILE2 | File1 and File2 have the same Inode numbers |
| FILE1 -nt FILE2 | File1 is newer (modification date) than File2 |
| FILE1 -ot FILE2 | File1 is older than File2 |
| -b FILE | File exists and is block special |
| -c FILE | File exists and is character special |
| -d FILE | File exists and is a directory |
| -e FILE | File exists |
| -f FILE | File exists and is a regular file |
| -g FILE | File exists and is Set-Group-ID |
| -G FILE | File exists and is owned by the Effective Group ID |
| -k FILE | File exists and has its Sticky Bit set |
| -L FILE | File exists and is a symbolic link |
| -O FILE | File exists and is owned by the Effective user ID |
| -p FILE | File exists and is a named pipe |
| -r FILE | File exists and is readable |
| -s FILE | File exists and has a size greater than zero |
| -S FILE | File exists and is a socket |
| -t [FD] | File Descriptor FD (stdout by default) is opened on a terminal |
| -u FILE | File exists and its Set-User-ID is set |
| -w FILE | File exists and is writeable |
| -x FILE | File exists and is executable |


© Dennis Rice

When evaluating a combined value in a shell, the parentheses must be escaped (must be preceded by a backslash).

## 9.38 TIFF Image Information

A TIFF image contains information that may be useful when manipulating the image. This ability may be important to the user that is working with TIFF images on various projects. Several utilities are available.

### 9.38.1 TIFF Information

Each TIFF image, or also known as a directory, contains information regarding the image. This may be displayed with the following command:

**$ tiffinfo [options] image.tif**

Options include:

| | |
|---|---|
| **-c** | Display the colormap and color / gray response curves, if present |
| **-D** | In addition to displaying the directory tags, read and decompress all the data in each image (but not display it). |
| **-d** | In addition to displaying the directory tags, print each byte of decompressed data in hexadecimal. |
| **-j** | Display any JPEG related tags that are present. |
| **-o** | Set the initial TIFF directory according to the specified file offset. The file offset may be specified using the usual C-style syntax. |
| **-s** | Display the offsets and byte counts for each data strip in a directory. |
| **-z** | Enable strip chopping when reading image data. |
| **-#** | Set the initial FIFF directory to # |

### 9.38.2 TIFFDUMP

The utility **tiffdump** displays information from files created according to the Tag Image File Format. The header of each TIFF file contains three values, Magic Number, Version, and First Directory Offset. These values are displayed, followed by the tag contents of each directory in the file. For each tag, the name, datatype, count, and value(s) is displayed.

When the symbolic name for a tag or datatype is known, the symbolic name is displayed followed by it's numeric (decimal) value. Tag values are displayed enclosed in "<>" characters immediately preceded by the value of the count field.

For example, and ImageWidth tag might be displayed as

ImageWidth (256) SHORT (3) 1<800>

**tiffdump** is particularly useful for investigating the contents of TIFF files that libtiff does not understand.


© Dennis Rice

## 9.39 Utility Time

When running a command or utility, it may be appropriate to know how long the command took to perform its task. The time can be monitored by using the command:

$ **time command**

The output displays the command, user and cpu statistical times.

```
$ time sleep 5
real    0m5.011s
user    0m0.000s
sys     0m0.010s
```

## 9.40 Wordwrapping Text

Quite often, you have data that is wider than the screen. Some display applications will automatically wordwrap the longer line, but others do not. If you desire to have a file that is formatted for screen output, issue the command:

$ **fold [options] file**

The output is issued to the stdout or monitor. The most common option is **–w**, which specifies the character width of the display, the default is 80 characters.

```
$ cat test.txt                          This is the test file
123123456456789789123456789123456789321665498732165498736925814736925814797136428597456136987412589631255789314569
25814736925814797136428597456136987412589631255789314569
```

```
$ fold -w 60 test.txt
123123456456789789123456789123456789321665498732165498736925
814736925814797136428597456136987412589631255789314569
```

```
$ fold -w 40 test.txt
1231234564567897891234567891234567893216
6549873216549873692581473692581479713642
8597456136987412589631255789314569
```

## 9.41 Determining the User ID

During the administration of a system, it is sometimes necessary to know the specific details about a user. A user may be having problems during their operation, and you need to assist them, and need to know some vital facts of their login.

Issuing the command **id** provides this information.

$ **id [user]**

```
uid=505(jdoe) gid=505(jdoe) groups=510(sales),580(dd)
```

The [user] option allows you to specify another user.

This tells us the following information:

uid: 505 Every user is assigned a numeric id.


© Dennis Rice

gid: 505 Every user also has a group created with the same name as their userid. In general, the groupid will also be the same number. Do not confuse these two entities or numbers, they are totally different, but inter-related.

Groups 510, 580:
These are additional groups that jdoe belongs to.

## 9.42 User Identity Information

Every user on a system has additional information that may be added. This information is optional.

The first option is the passwd comment field. Quite often, we fill this field with the user's name. Because it is a comment field, any information may be placed in it. To modify the contents, issue the command:

$ **usermod –c "user's name" username**

For the second option, we utilize the properties of **finger**. When you finger someone, you requested user information about them. To specify the finger information, issue the command:

**$ chfn username**
***Name:*** **User's Name**
***Office:*** **User's (Work) Address**
***Office Phone:*** **User's Office Phone**
***Home Phone:*** **User's Home Phone**

This updates the information for the user. The updates are stored in the /etc/passwd file.

To read the finger information for a user, we issue the command:

**$ finger username**

The response will be:

***Login: username*** ***Name: User's Name***
***Directory: /home/username*** ***Shell: /bin/bash***
***Office: Office Address*** ***Office Phone: Phone***
***Home Phone: Home Phone***
***Last Logged In***
***Mail Status***
***No Plan***

This process also stores the information in the /etc/passwd file in the comment field. You can open the file and read the contents for the specified record.

In order to allow the information to be observed, we must set up your system as a finger server. First check to see if it is active by using the chkconfig utility. If set to off, then turn it on. Activate and restart the service. If you do not activate the service, you will be able to only obtain data from your system.


© Dennis Rice

Under normal circumstances, we do not desire to activate the finger service, as it may be used to obtain information about users. This compromises your system security.

## 9.43 History of Past Commands

Over a period of time, we issue many commands on our system. Quite often we tend to repeat the commands a number of time – especially if we are doing something that is repetitive.

Unix and Linux maintain a list of the last commands issued. The list may be set by the user, but by default is either 500 or 1000 lines long. There are several options for accessing the list and using past commands.

### 9.43.1 Listing the Previous Commands using history

The first command available for listing previously issued commands is:

**$ history**

This will present a numbered list, one command per line. If you have been issuing commands for some time, then your list is quite long and you will see them scroll past you on the screen.

Utilizing the **pipe** ( **|** ), you can limit the display to only the top or bottom 10 commands by using it in conjunction with either the **top** or **tail** utility. If you need to select a specific command, you may utilize the **grep**.

To utilize one of the commands, several options are available.

1. Use the up or down arrow keys to select the desired command. Normally you will start with a blank line, using the up arrow will scroll through the list from the bottom up, or most recent issued command first. You can return to the bottom by using the down arrow key.
2. You can reissue a command by using the "!" (bang) command and the number of the previously issued command. This will bring up the command at the prompt, where upon hitting the ENTER key will cause the command to be issued.
3. You can issue the bang and the first couple of letters, where upon the command will be completed.

### 9.43.2 Listing Previous Commands using fc

The command **history** listed the last 1000 command that the user has issued – overwhelming. If we need to only look back through the last 20 commands, we can issue the command:

$ **fc -l**

This provides the same basic information, and a lot more manageable. See the previous discussion for more detail.

### 9.43.3 Completing a Filename using the TAB Key

Beyond the ability to recall past commands from the history list, you can also use the **TAB** key to complete a filename. By typing the first few characters of a


© Dennis Rice

filename and then pressing the TAB, the system will automatically complete the filename up to an unambiguous point. Pressing the TAB again will list all of the filenames that match the characters typed in so far.

## 9.44 Unattended Jobs using *at*

We often have the need to have a job or process take place at a later time. Common examples would be the scheduling of a job late at night after you have gone home. This enables the computer to perform a task when it otherwise would be operating relatively idle.

There are several options to perform this task. If a job is to be performed periodically, such as a network backup, then we would use the **cron** application – this will be covered in the next section.

For infrequent applications, we have two applications which might be utilized – **sleep** and **at**. **Sleep** can delay the process of a job, but does not allow one to still use the computer unless it is performed in the background. **At** allows one to schedule a job at a later time and then resume using the computer for their normal tasks. An excellent application for **at** would be the researching of a database for specific information that will require several hours, thus we might like to run the program late at night.

The output of the **at** process must be to a file. That is, either you must have the output directed to a file, or the application being processed must create a file. This is because the **at** process is not capable of directing the output to the standard output (monitor).

The generalized format of setting up the **at** process is as follows:

**$ at hhmm**

The hhmm is the hour and minute that you wish the process to take place, and may be in the format of either 12 hour, with an AM / PM designator, or 24 hour.

Following the first line, you need to input the command that you wish to have executed. Multiple commands may be implemented, one per line, or all on one if each is separated by a semicolon. Each command is entered following the "**at>**" prompt. After all of the commands have been entered, on a new prompt line, enter a CTRL D (^D).

As an example of creating a simple file to be performed in three minutes, creating a simple new file in the /lab directory.

**$ date**
**$ *8 Feb 2003 12:20:25***
We do the date just to find out the current time so we can have the command completed in a few minutes.
**$ at 1223**
***at*> echo "AT command 1" > /lab/at1**
***at*> ^D**
**$ job 7 at 2003 - 02 - 08 12:23**


© Dennis Rice

Now in several minutes, 12:23, our system will create the file /lab/at1 will be created with the contents of "AT command 1". The job has been assigned the numeric value of 7 in this example.

Several minutes later (i.e. after 12:23) we go to the /lab directory, and find the file 'at1'. Displaying the file, we will find that it contains the contents of "AT command 1".

So now we know how to have a command implemented at a later time, but how might we observe what jobs are in queue. Again we have two options:

**$ at –q** or
**$ atq**

This will generate a list of jobs that are pending to be processed.

An alternative to having the output of a process being directed to a file, one could send an email to either yourself or to someone else. This may be performed by issuing the command:

$ **at > echo "Linux class test at 10 AM." | mail your-username**

Now a mail will be sent to you (on the system you are working on) with the contents of 'Linux class test at 10 AM.' By putting in a full email address, and assuming that a mail system has been properly set up, one could send mail to whomever one would want.

## 9.45 Unattended Periodic Jobs using *cron*

As an administrator, we often require the need to repeat a specific command on a regular basis. A common requirement would be for a daily backup.

Once every minute, a process called **cron** is activated, checking to see if there are processes that need to be run. If one should exist, it runs that script or application. Running an application may be set up by the minute, hour, day of month, or day of week.

The default file that maintains this information is **/etc/crontab**. We can either list the file directly (less), or use the command **crontab –l**. The format of the output is as:

1) ***01 * * * * run-parts /etc/cron.hourly***
2) ***02 4 * * * run-parts /etc/cron.daily***
3) ***22 4 * * * run-parts /etc/cron.weekly***
4) ***42 4 1 * * run-parts /etc/cron.monthly***

We read these command lines as:

**minute hour day-of-month month day-of-week /path/command [command-options]**

The range of values for each of these fields is:

| | | |
|---|---|---|
| minute | 0 – 59 | |
| hour | 0 – 23 | |
| day-of-month | 1 – 31 | |
| month | 1 – 12 | |
| day-of-week | 0 – 6 | where 0 = Sunday and 6 = Saturday |


© Dennis Rice

A "*****" is a wild card – all values are accepted. A range of values may also be specified, such as 1 – 5.

The above four cron commands generate the following:

- At 1 minute past every hour, the application **run-parts** runs all scripts in the **/etc/cron.hourly** directory.
- On every day at 4:02 AM, the application **run-parts** runs all scripts in the **/etc/cron.daily** directory.
- On every week at 4:22 AM, the application **run-parts** runs all scripts in the **/etc/cron.weekly** directory.
- On every month on the first day of the month at 4:42 AM, the application **run-parts** runs all scripts in the **/etc/cron.monthly** directory.

To run a script (application) at one of these specified times, all we need to do is to create a script in the appropriate directory.

If we want to create or modify the user crontab file, we need to edit the crontab file. This file should not be edited manually, but be modified using the command **crontab X**, where X is:

– l lists the existing contents of the crontab file
– r remove an entry from the crontab file
– e edit the crontab file

In doing a listing (ls) of cron*, we observe four cron lines – cron.hourly, cron.daily, cron.weekly, and cron.monthly. If we want to create a new entry, for example to ping another station to verify its activity, we could have the command issued once every 2 hours during the business day at 45 minutes past the hour. Our time codes need to be:

45 8,10,12,14,16,18 $*_1$ $*_2$ 1–5

| | |
|---|---|
| 45 | minute of the hour |
| 8,10,12,14,16,18 | hours |
| $*_1$ | every day of the month |
| $*_2$ | every month of the year |
| 1–5 | every Monday through Friday |

***Special Note:***

The default editor is **vi**. When crontab –e opens the file for editing, you must be able to navigate to edit it. The following commands will get you through the editing with minimal effort:

| | |
|---|---|
| i | puts vi into insert (text) mode |
| esc | puts vi into the command mode |
| : | at command mode, tell it that a command will follow |
| w | at command mode, writes the file out |
| q | at command mode, quits vi |

The command that we need is **ping –c 5 ahost**, where **ahost** specifies an IP address obtained from our **/etc/hosts** file.


© Dennis Rice

Using **crontab –e**, we enter the following information:

45 8,10,12,14,16,18 * * 1–5 ping –c 5 ahost

This entry will cause a ping to be transmitted 5 times to the ahost system (assumed to have an entry in the /etc/hosts file) to be issued at 8:45 AM, 10:45 AM, 12:45 PM, 2:45 PM, 4:45 PM, and 6:45 PM on Monday through Friday for every week of every month.

Upon saving the file, it creates a new (or edits the existing) file **/var/spool/cron/username**. If you are logged in as ***root***, then the file name will be **/var/spool/cron/root**.

When the command is run, its output is not to the screen, but is sent as an email to the user, in our case to ***root@localhost.com*** . To read the mail, issue the command **mail**. This will bring up a listing of your existing mail message – now hit the **1** key to read the first message; if there are multiple messages, you can either hit the number or the **n** key (next).

Now say you want to create a "very simple" script to demonstrate the operation. First change to the **/root** directory, then using nano, create a new file called **echohi**. In this file, enter the following lines:

```
#!/bin/bash
#  echo the following line
echo  "Hello there, have a nice day."
```

Before we can run the script, we need to make it executable. To do this, we need to modify the permissions of the file. Issue the command:

**$ chmod +x echohi**

Going back and looking at the attributes of the **echohi** file, you will observe:

**-rwxr-xr-x 1 root root …**

You can test this script file by issuing the command:

$ **/root/echohi** or

$ **./echohi** (if in the /root directory)

Note that in order to execute a command from within a directory, one must tell the system where to find it – thus the reason for the " **./** " in front of the command. This tells the system that it can find the command within the immediate directory.

And you should observe the following text:

***Hello there, have a nice day.***

Now again edit the crontab file (**crontab –e )** and insert the following line:

**0,10,20,30,40,50 10,11,12,13 * * * /root/echohi >> /root/echohifile**

Remember to enter into the text mode to insert the character string, and when done, to write / save the file, then exit. Once the file has been modified, it will immediately start acting on the command.


© Dennis Rice

## 9.46 Creating Your Own Command

We often want to simplify commands that we use continuously. This might be an advantage where we can combine several commands together and rename it to a unique name.

To observe existing alias commands, issue the command: **alias**

A new alias can be created by typing the command:

**$ alias {newname}='{command line}'**

Note that the single quote character " ' " must be included and that there are no spaces either before or after the equal sign.

A simple example is if you wish to list the details of all files, which is **ls -1.** Say you want to use an alternative command of **lsl** to mean the same. Give the command:

**$ alias lsl='ls –l'** {no spaces around "**=**" sign}

Unfortunately with this command format, it is only good during the specific power-up of Linux if you shut down, the alias command will be lost. To maintain the command every time you power up, you need to modify a file in the user home directory using the vi (pico) editor.

If you wish to make changes which are exclusively for you, then you need to modify the **/root/.bash_profile** in your home directory (or the home directory of the logged on user). We first need to change to the home directory, type:

**$ cd**

this returns you to your home directory. If you are the root administrator, you will be at **root,** otherwise you will be at the appropriate user's home directory.

Using **vi** or **nano,** type:

**$ vi .bash_profile** make sure you insert the **"."** before the bash.

or

**$ nano .bash_profile**

If using vi, change from the command line mode to the insert mode (type i) and add the line just below the first set of comments:

**alias lsl='ls –l'** "l" is the lower case "L".

Before these become active, the user must either

logout and then login again or

shutdown and reboot

From the command line you can now type the command **lsl** and you will receive a list of all the files with their respective properties.

If you wish to create alias which may be used by everyone, you need to perform the same process to the **/etc/bashrc** file, add the following lines to the **/etc/bashrc** file:

**alias lsl='ls –l'** (no spaces around "=")

**alias lsa='ls –la'** lists all files and directories

**alias lsd='ls –d'** lists only directories


© Dennis Rice

**alias cls='clear'** DOS command to clear the screen

After saving the file, log out and back in to make them effective.

You can enter an alias on the fly, only active during your present login session by immediately issuing the alias='xxxx' command. Remember to not have spaces around the equal sign.

If you desire to remove an alias command, issue the reverse command:

$ **unalias alias-command**

## 9.47 Obtaining File Information

The File utility provides you with information of what type of file you are working with. Three different attributes are tested – filesystem, magic number, and language.

The syntax of the command is:

$ **file filename**

The attributes are:

| | |
|---|---|
| Filesystem | Directory, File, Link |
| Contents | Character type |
| | ASCII text |
| | Binary |
| | Executable |
| | Data |
| | Empty |
| Magic Number | Checks to see if a file contains some form of format. |
| Language | Checks to see if a file contains a specific character type. |

The best way to understand this utility is through the lab examples.

## 9.48 File Types [2]

Previous discussion with files has been with files that held user information. Recall that we previously stated that Unix and Linux treat everything as a file, including hardware devices. Here we need to review additional file types.

### 9.48.1 Normal Files

In review, we have observed the following file types:

| | |
|---|---|
| - | Standard file carrying data |
| d | Directory file, carrying a list of other files |
| l | Link file, connecting two files together |

### 9.48.2 Block Devices

Now we will expand the file types to include:

[2] Linux Administration, A Beginner's Guide, 2nd Ed, Steve Shah, Osborne McGraw-Hill


© Dennis Rice

b Block device. This basically device drivers. The system transfers a normal file ( -, d, l) to a hard (floppy) drive. In this case, our normal file is transferred to another file type.

c Character device. This is used to transfer data from a normal file to an input / output device, such as a serial interface (communications port or USB port). Here characters are transferred between a normal file and an I/O device.

P Named Pipe. A named pipe allows for system inter-process communications. It is a special file type that takes input from one file type and passes it on to another file type. This type of file type may be necessary when a file is not able to accept data input from the stdin (keyboard) device.

### 9.48.3 Listing Block Devices Attributes

When we previously listed the attributes of a file we used the **ls –l** (**ll**) utility. We still do, but the output is slightly different. Most often, block devices will be found in the **/dev** directory.

**$ ll /dev/hda**

***brw-rw- - - 1 root disk 3, 0 Jan 1 2004 /dev/hda***

This has a slightly different format than previously noted. Obviously it starts with a “b”, indicating a block device. We then observe that it is owned by root, but the group name is “disk”. When doing a listing of all “hda” devices one will observe that drive “a” can support over 30 partitions, with the drive being device “ 3, 0 ”. The first value is the major number and the second is the minor. In review of the listings, one can learn that:

Primary Drive Interface
Secondary Drive Interface
SCSI Controller 1
SCSI Controller 2
Floppy Drive
ttyS Interface (Comm Port)

The minor number indicates the partition number in the case of a hard drive, or a floppy density in the case of a floppy drive.

## 9.49 Midnight Commander

There is a command in Linux similar to an old DOS function developed by Peter Norton that provided enhanced abilities to view the various files in a tree structure. Today this format is provided in the Windows Explorer function, where there are two windows, the left showing the directories and the right showing directories (folders) and files within the specified directory.

This process is called **midnight command,** and is initiated with the command:

**$ mc**


© Dennis Rice

From within **mc** you can view the directory structure of your Linux system, contents of directories, execute a file, and open text files for viewing.

The screen displays two separate screens – left and right. Each represents an independent view of your location, that on startup show the same directory. By default, you are located in the left hand screen, but can change to the right hand screen by clicking the TAB key. You may switch back and forth using this technique.

Using the arrow keys, you are able to move to the different directories / files. After selecting a directory or file, you can utilize the Function Keys to perform desired actions.

The menu at the bottom of the screen is for the Function Keys. These are assigned at the top level as:

1. F1 Help
2. F2 Menu
3. F3 View
4. F4 Edit
5. F5 Copy
6. F6 Rename / Move
7. F7 Make Directory
8. F8 Delete
9. F9 Pull Down
10. F10 Quit

Sub menus may be available from several of the above, which will cause the definition of the F-keys to change at the bottom of the screen.

## 9.50 DOS Mtools

One of the features of Linux and Unix is the ability to access and read the various Windows file systems. The access of file systems that are part of a hard drive will be covered at a latter time, but we often transfer files via a floppy drive.

To work with this lab, you will need a floppy disk formatted under a DOS system.

Normally we need to "mount" a removable drive, such as a zip or cdrom drive disk. DOS floppy disks do not need to be mounted, but may be accesses through a special set of commands called "**Mtools**". They are applicable in general to any MS-DOS file system, but for this discussion we will be working with floppy disks.

Thus we are able to manipulate a DOS system. Note that with these commands we can create a directory and save files to it, but we are not able to edit a file within a DOS directory – for this we must transfer the file to our Linux / Unix system, edit it, then return it to the DOS system.

Several of the more used commands include:

### 9.50.1 mcopy Utility

To copy a file from a DOS disk to your hard drive, issue the command:

**$ mcopy a:filename /path/filename** or
**$ mcopy a:filename filename** (if in the desired directory)


© Dennis Rice

To copy a file from Linux to a DOS floppy, issue the command
**mcopy /Linux-path/filename a:/dos-path/filename**

If the file on the DOS disk is at the top of the directory structure, then the dos-path is not required. If you are located in the directory where you wish to place the file, again the Linux-path is not required.

### 9.50.2 mdir Utility

To list the contents of a DOS floppy, you need to only issue the command:
**$ mdir**

Linux will automatically assume that you mean Floppy Drive a. If you should have two floppy drives, then you will need to specify the drive.

### 9.50.3 mtype Utility

Like the less command, we are able to display the contents of a file that resides on a floppy disk. We need to issue the command:
**$ mtype /dos-path/filename**

The following is a summary of most of the mtool commands. Your are not able to read a DOS disk with the normal Unix / Linux commands.
A partial list of available commands includes:

| m command | DOS command | Function |
|---|---|---|
| mattrib | attrib | changes attribute flags |
| mcd | cd | changes DOS directory |
| mcopy | copy | copies files from DOS to Linux |
| mdel | del/ erase | deletes a DOS file |
| mdir | dir | displays contents of DOS directory |
| mformat | format | formats a high density floppy disk |
| mlabel | label | writes a label to a floppy disk |
| mmd | md / mkdir | creates a DOS directory |
| mrd | rd / rmdir | deletes a DOS directory |
| mread | copy | copies files from DOS to Linux |
| mren | ren / rename | renames a DOS file |
| mtype | type | displays contents of a DOS file |
| mwrite | copy | copies a Linux file to DOS |

The user should investigate the other commands.
Normally, these commands are most often used on a DOS floppy, but may also be used on an attached DOS hard drive.
In general, the command syntax will be of the form;
**$ mcommand filename / action**

When using the any of these commands, refer to the appropriate **man** or info pages for proper syntax.


© Dennis Rice

## 9.51 Compression Techniques (Not Complete)

Linux offers a wide variety of compression techniques in order to make a file, or a group of files smaller in size. Various algorithms are available, thus the variety of techniques. Here we will review several of the more popular techniques.

### 9.51.1 compress / uncompress:

Original Unix compression algorithm. Not highly supported as it was originally patented. File extension is typically '.z' .

### 9.51.2 gzip / gunzip:

Replaced compress that is patent free (open-source). Latest versions support the compress algorithm. File extension is typically '.gz' .

Gzip is not the same as the MS Windows zip format, although winzip is able to uncompress '.gz' files.

Syntax is:

**$ gzip –[options] file(s) > gz-filename**

Options (others exist):

| | |
|---|---|
| -c | compress |
| -d | dcompress (same as gunzip) |
| -9 | highest compression |
| -1 | fastest compression (numeric one) |
| -l | list details of the gzip file (lower case L) |
| -r | recursive – compress all subdirectories and files below the present directory |

Note that in compressing the file, according to the man page, the original file is deleted – but this was observed to be in error. Normal compression goes to the stdout (screen) unless the redirect is specified.

Example:

**$ gzip –c cover > cover.gz**

### 9.51.3 zip / unzip:

A compression algorithm that is compatible with the PKWARE ZIP function or WINZIP. One is able to add, expand, list, or test zip files.

Syntax is:

$ **zip –[options] zip-filename files**

Options (others exist):

| | |
|---|---|
| -@ | read input names from the stdin (keyboard) |
| -*n* | numeric value, specifies compression value, 1= fastest, 9= highest |
| -b path | specify a path for temp files |
| -d | delete entries in the zipped file |
| -e | encrypt compression |
| -f | replace changed files |
| -g | grow, or append to an existing zipped file |


© Dennis Rice

**zip** is used to compress files, unzip is used to unzip the compressed file.
Example:

**$ zip cover.zip cover**

### 9.51.4 bzip2

**bzip** allows the compression of a file or files with block sorting.
Syntax is:

**$ bzip2 –[options] file(s)**

Options (others exist):

-n sets block size, 1= 100K, 9= 900K
-c compress to the stdout (monitor)
-d decompress
-k keep input files

Example:

bzip2 -c cover > cover.bz2

## 9.52 Backing Up Files (Not Complete)

One of the most important tasks that is most often neglected is the backing up of our data. Originally all backups were made to a magnetic tape media, being archived in their basic format without compression. (You might recall that computer magnetic tape reels that were 12 inches in diameter.) Hence we have a Tape Archive – or **TAR**.

Many files today are maintained in a TAR format. These files today may be found on the Internet.

The biggest draw back to TAR as previously mentioned is that the files are uncompressed. This limitation has subsequently been overcome by allowing an option which provides compression using the **gzip** algorithm.

The general format of the command is:

**$ tar *options*f *archive-name*.tar *path/filename***

### 9.54.1 tar Example[3]

TAR, or Tape Archiver, provides a means to archive files for backup or transfer. Although it was originally designed to store data to a tape drive, it now stores data to a hard drive. Here we will create a small TAR file and then expand it back to it original set of files.

First we will create a small script file that will be used to create our tar files. From the **/labs** directory, create the file **/labs/tarbkup** with the following contents:

**/etc/passwd**
**/etc/shadow** You may create a file with whatever
**/etc/group** contents you want.

[3] Setting Up a Linux Internet Server; Coriolis – Visual Black Book; Tsuji, Watanabe; ISBN 1-57610-569-5


© Dennis Rice

**/etc/hosts**
**/etc/named.conf**

Save the file and exit. Make sure you include the full path on each line for the file you wish to save.

Now we wish to create our tar file, issue the command:

**tar cvfzT /lab/etcbkup.tar.gz /lab/tarbkup**

The tar command is one of the few that does not require the " – " in front of the options list. From experience, it has been found that using the " – " may cause the command to fail.

The options provide for the following:

- **c** Creates a new archive file
- **v** Displays archive process (verbose)
- **f** Specifies the archive filename (**etcbkup.tar.gz)**
- **p** Preserve file permissions
- **z** Compress the total file for optimum storage
- **T** Get files to archive from specified file (**tarbkup**)

Additional options include:

- **A** Concatenate, or append tar files to an archive
- **d** Find the differences between two archives
- **r** Append files to the end of an existing archive
- **t** List the contents of an existing archive
- **u** Update only those files in an archive that are newer that the one in the archive
- **j** Use the bzip2 utility for compression of an archive
- **k** Keep old files, do not overwrite existing files from the archive
- **N** Include files in an archive that were created after specified date
- **v** Provide a verbose output of the command process
- **Z** Use the compress utility for compression of an archive

Several items must be noted. First we need to specify the **f** option and the file name, otherwise tar will attempt to store the file on the tape drive – which does not exist. We created a new file **/lab/etcbkup.tar.gz** using the script list file **/lab/tarbkup**. The **.tar** tells us that we used the tar process, and that the file is compressed, indicated by the **.gz**. The .tar and .gz are only necessary for the user, as the process does not care what the file name is (it's a human thing). Second, the fully qualified path name must be specified for both files.

Now lets restore (unpack) our file. Issue the command:

**$ tar xvpfz etcbkup.tar.gz**

where:

- **x** Extract files from the tar file
- p Preserve file permissions
  Other options are the same

If your original files should become corrupted, you will now be able to copy them back to their original location using the **cp** command.


© Dennis Rice

To list the contents of an archive, issue the command:

```
$ tar tf etcbkup.tar
etc/passwd
etc/shadow
etc/group
etc/group
etc/hosts

$ tar tvf etcbkup.tar
-rw-r--r-- root/root      1124 2006-06-30 18:53:27 etc/passwd
-rw-r----- root/shadow     577 2006-07-01 08:48:16 etc/shadow
-rw-r--r-- root/root       475 2006-06-30 18:30:04 etc/group
-rw-r--r-- root/root       475 2006-06-30 18:30:04 etc/group
-rw-r--r-- root/root       681 2006-06-30 18:56:18 etc/hosts
```

Additional options exist to the **tar** command. Check out the **man tar** manual page to review what is available.

## 9.53 Quotation Marks (Not Complete)

If you look at your keyboard closely, you will find three different types of quotation marks. In the Unix / Linux world, these may have different meanings, depending where and how they are used.

The three quotation marks are:

| | | |
|---|---|---|
| Single Quote | ‘ | (immediately left of ENTER key – lower case) |
| Double Quote | “ | (immediately left of ENTER key – upper case) |
| Back Quote or Back Tic | ` | (immediately left of **1** key – lower case) |

When working at the Command Line Interface, there are generally no difference between the Single Quote and Double Quote. Differences become significant when we enter into the various programming languages. Bash, Perl, Python and other languages interpret variables differently when using either the Single of Double quote. For example:

```
$OS = “Linux”
echo “The OS is $OS.”     ->     The OS is Linux.
echo ‘The OS is $OS.’     ->     The OS is $OS.
```

The Back quote is used to execute a command within another command. If you wish to learn to do scripting, you need to do additional self-study.

A simple rule of implementation of quotes is:

Single quotes are used inside of double quotes.

## 9.54 Suspend Execution

On occasion it might be necessary to temporarily suspend the operation of the system. The command to do this is **sleep**. This pauses execution of the shell for the number of specified seconds. The command is:


© Dennis Rice

**$ sleep n**

where **n** is in seconds.

## 9.55 System Uptime

While running various applications, it is sometimes advantages to learn how long a system has been operational. To learn this, issue the command:

**$ uptime**

***10:43pm up 4 days 11:07, 1 user, load average: 0.00, 0.00, 0.00***

This command will typically be used within a script.

## 9.56 Commands Used in this Chapter

| | |
|---|---|
| alias | Creates a new command for another command |
| at | Utility to issue a command at a later time |
| batch | Runs a file of other commands |
| bzip2 | Compresses or uncompresses a file |
| cal | Displays a calendar |
| cat | Displays the contents of a file |
| cd | Changes directory |
| chfn | Allows one to modify the comment portion of a user in the passwd file |
| chmod | Utility to change the permissions of a file |
| cksum | Creates a hash of a file for verification purposes |
| cmp | Compares two files character by character, listing the differences |
| colrm | Removes columns from a file |
| column | Creates columns from data |
| comm | Displays the difference of two files |
| compress | Compresses a file |
| cron | Utility to issue a command on a periodic basis |
| crontab | Utility for configuring a cron process |
| csplit | Splits a file based on the specified number of lines |
| cut | Extracts a specified range of characters from a file |
| date | Displays or sets the system date and clock |
| dc | Application to provide a precision calculator |
| diff | Compares two files line by line, listing the differences |
| dir | Displays the contents of a directory |
| echo | Displays on the stdout (monitor) the specified message |
| enscript | Converts a text file to postscript format |
| expand | Converts tabs to spaces |
| export | Writes a value to a user's environment |
| factors | Displays the prime factors of a number |
| fc | Lists a portion of the history file |
| file | Displays the file type |
| finger | Displays the comments portion of a user from the passwd file |


© Dennis Rice

| | |
|---|---|
| fc | Displays a limited portion of the history file |
| fold | Provides a means to word-wrap lines of a file |
| gpasswd | Creates a group password |
| grep | Searches a file or directory for a string |
| gunzip | Uncompresses a file |
| gzip | Compresses a file |
| history | Displays the last 1000 commands issued |
| id | Displays the user's id value |
| info | Displays a commands properties |
| ispell | Displays mis-spelled words one by one and suggests correction |
| job | Displays the number for the jobs in process |
| login | Creates a new login screen |
| logname | Displays the user login name |
| look | Looks up word in dictionary for spelling verification |
| ls | Displays the contents of a directory |
| mattrib | DOS floppy display attributes command |
| mcd | DOS floppy change directory command |
| mc | Midnight Commander utility |
| mcopy | DOS floppy copy command |
| mdel | DOS floppy delete file command |
| mdir | DOS floppy display directory contents command |
| md5sum | Creates a hash of a file for verification purposes |
| merge | Combines multiple files together |
| mev | Detects mouse movement |
| mformat | DOS floppy format command |
| mlabel | DOS floppy to write a label to a floppy disk |
| mmd | DOS floppy to create a new directory |
| mrd | DOS floppy to remove an existing directory |
| mread | DOS floppy to display a file's contents |
| mren | DOS floppy rename a file or directory |
| mtype | DOS floppy to display file's contents |
| mwrite | DOS floppy to write information to a floppy |
| mzip | Utility to extract a zip disk from the zip drive |
| nano | Line editor |
| newgrp | Creates a new group for users |
| nice | Changes the priority of an application |
| nl | Numbers the lines of a file |
| paste | Combines two files side by side |
| pico | Line editor |
| pipe | Directs the output from one command into another |
| pr | |
| rev | Displays the text in reverse order |
| script | Logs to a file the output of a command sequence |
| sleep | Pauses the processing of a command for a specified period of time |
| sort | Sorts a list of items |


© Dennis Rice

| | |
|---|---|
| source | |
| spell | Displays mis-spelled words in a file |
| split | Splits a file based on length |
| ssh | Secure Shell utility |
| statserial | Displays the status of the serial pins of a serial port |
| TAB | When used while entering a file name, completes the file name |
| tail | Displays the last 10 lines of a file |
| tar | Archives a file for storage |
| tee | Allows command output to be both saved to a file and output to the stdout (monitor) |
| test | Provides a means to test various conditions |
| tiffdump | Displays data regarding a tiff image file |
| tiffinfo | Displays information about a tiff image file |
| time | Displays the amount of time that an application takes |
| top | Displays the top 10 lines of a file |
| tty | Displays the terminal that one is logged onto |
| unalias | Removes an alias command |
| uname | Displays the attributes of a system |
| uncompress | Uncompresses a file |
| usermod | Allows one to modify attributes of a user |
| unzip | Uncompresses a file |
| uptime | Displays the time a system has been operational |
| wc | Counts the number of characters, lines and words in a file |
| xhost | Allows one to remote an X application to a remote terminal |
| vi | Line editor |
| zip | Compresses a file |

## 9.57 Chapter Review Questions

1. You need to check the spelling of a word. What command is issued?
   a. spell
   b. dict
   c. look
   d. check
2. A task needs to be issued just once at a specific time. What command is used?
   a. at
   b. cron
   c. issue
   d. run


© Dennis Rice

3. You wish to archive and compress using gzip a file, what command is used?
   a. compress
   b. tar
   c. tar -cvzf
   d. zip
4. You observe that the system time is not correct. What command is issued to correct it?
   a. time
   b. day
   c. year
   d. date
5. A downloaded file needs to be verified using the md5 check. What command is used?
   a. cksum
   b. crc
   c. md5sum
   d. verify
6. You need to split a file by length so that it will pass through an ISP's email system. What command is used?
   a. divide
   b. length
   c. sep
   d. split
7. During a telnet session you wish to record all commands and responses. What command is used?
   a. telnet | script
   b. telnet | script filename
   c. telnet | tee
   d. telnet | tee filename
8. Within a script, you need to display a calendar for a specific month and year. What command is issued?
   a. cal ‘year’
   b. cal ‘month’
   c. cal ‘month’ ‘year’
   d. cal
9. You want to have fun with a friend by sending a file that is reversed. What command is used?
   a. back
   b. galello
   c. invert
   d. rev


© Dennis Rice

10. You need to capture the year to a variable within a script. What command is issued?
    a. day + % y
    b. date + % y
    c. day %y
    d. date %y
11. You need to check a file for just spelling. What command is used?
    a. ispell
    b. look
    c. spell
    d. ten
12. You have generated a report from a database and need to extract characters in columns 13 through 27. What command is used?
    a. column –b 13 – 27
    b. col –b 13 – 27
    c. cut –b 13 – 27
    d. pull –b 13 – 27
13. The output of a command must be directed to both a file and the screen. What command is used?
    a. log
    b. recall
    c. script
    d. tee
14. You are continually issuing a rather long command, what could be done to improve the process?
    a. create a command
    b. create a name
    c. create a process
    d. create an alias
15. You have written a script and need to know how many lines are in the file. What command is issued?
    a. lines
    b. wc
    c. wc –l
    d. wc –s
16. The lines of a file need to be numbered, including blank lines. What command is used?
    a. line
    b. nl
    c. nl -ba
    d. number


© Dennis Rice

17. You wish to check a file for spelling, along with suggestions. What command is used?
    a. fix
    b. ispell
    c. look
    d. spell
18. You need to display the contents of a variable to the stdout. What command is issued?
    a. echo VARIABLE
    b. echo $Variable
    c. echo $VARIABLE
    d. echo
19. Your math teacher wants to know the prime numbers of another number. What command is used?
    a. base
    b. factor
    c. number
    d. prime
20. A task needs to be issued on a periodic basis. What command is used?
    a. at
    b. cron
    c. run
    d. timer
21. You have just received a list of data from a device and need to sort the data. What command is issued?
    a. sort
    b. order
    c. list
    d. ls –o
22. A file is created to contain a list of commands. What command is used to run the file of commands?
    a. batch
    b. execute
    c. run
    d. source
23. You need to search a file for miss-spelled words and have suggestions made for corrections. What command is issued?
    a. spell
    b. look
    c. ispell
    d. correct


© Dennis Rice

24. In writing a script, you need to test a condition. What command is used?
    a. check
    b. make
    c. query
    d. test
25. You wish to login under a different name. What command is used?
    a. exit
    b. login
    c. logon
    d. open
26. What set of commands allow one to directly access a DOS formatted floppy drive?
    a. DOStools
    b. mtools
    c. Wintools
    d. Xtools
27. You have received a list of data and need to format it into columns. What command is issued?
    a. cal
    b. col
    c. column
    d. column 4
28. You need to check out the spelling of different words. What command is used?
    a. dictionary
    b. ispell
    c. look
    d. spell
29. You have logged into a remote system and need to know which terminal you are on. What command is used?
    a. console
    b. term
    c. terminal
    d. tty
30. You need to research a word from the system dictionary. What command is issued?
    a. spell
    b. list
    c. look
    d. ispell
31. You run a configuration script for configuring an application, what utility is used to save the answer / question process?
    a. listen
    b. script
    c. tee
    d. write


© Dennis Rice

32. What command is used to modify an application's priority?
    a. level
    b. nice
    c. priority
    d. set
33. Which command provides a report of the first error found when comparing two files?
    a. cmp
    b. compare
    c. diff
    d. rpt
34. You have two files where you need to combine the data side by side. What command is issued?
    a. link file1 file2 > file3
    b. tie file1 file2 > file3
    c. sbs file1 file2 > file3
    d. paste file1 file2 > file3


© Dennis Rice

**Chapter Index**




© Dennis Rice

© Dennis Rice

© Dennis Rice