$ cal -- show calendar

### $ passwd --Change Password

### Current pwd?

### New pwd?

### Re Type?

### Who Are You?

$ whoami

### Who is Logged in?

$ users

$**logout**

**halt**

Brings the system down immediately

**init 0**

Powers off the system using predefined scripts to synchronize and clean up the system prior to shutting down

**init 6**

Reboots the system by shutting it down completely and then restarting it

**poweroff**

Shuts down the system by powering off

**reboot**

Reboots the system

**shutdown**

Shuts down the system

## Listing Files

To list the files and directories stored in the current directory, use the following command −

$ls

Here is the sample output of the above command −

$ls

bin hosts lib res.03

ch07 hw1 pub test\_results

ch07.bak hw2 res.01 users

docs hw3 res.02 work

The command **ls** supports the **-l** option which would help you to get more information about the listed files −

$ls -l

total 1962188

drwxrwxr-x 2 amrood amrood 4096 Dec 25 09:59 uml

-rw-rw-r-- 1 amrood amrood 5341 Dec 25 08:38 uml.jpg

drwxr-xr-x 2 amrood amrood 4096 Feb 15 2006 univ

drwxr-xr-x 2 root root 4096 Dec 9 2007 urlspedia

-rw-r--r-- 1 root root 276480 Dec 9 2007 urlspedia.tar

drwxr-xr-x 8 root root 4096 Nov 25 2007 usr

drwxr-xr-x 2 200 300 4096 Nov 25 2007 webthumb-1.01

-rwxr-xr-x 1 root root 3192 Nov 25 2007 webthumb.php

-rw-rw-r-- 1 amrood amrood 20480 Nov 25 2007 webthumb.tar

-rw-rw-r-- 1 amrood amrood 5654 Aug 9 2007 yourfile.mid

-rw-rw-r-- 1 amrood amrood 166255 Aug 9 2007 yourfile.swf

drwxr-xr-x 11 amrood amrood 4096 May 29 2007 zlib-1.2.3

d – directory rwx(user) r-x(group) r-x(other)

r :read

w : write

x : executable permission

- : no access

$

Here is the information about all the listed columns −

* **First Column** − Represents the file type and the permission given on the file. Below is the description of all type of files.
* **Second Column** − Represents the number of memory blocks taken by the file or directory.
* **Third Column** − Represents the owner of the file. This is the Unix user who created this file.
* **Fourth Column** − Represents the group of the owner. Every Unix user will have an associated group.
* **Fifth Column** − Represents the file size in bytes.
* **Sixth Column** − Represents the date and the time when this file was created or modified for the last time.
* **Seventh Column** − Represents the file or the directory name.

In the **ls -l** listing example, every file line begins with a **d**, **-**, or **l**. These characters indicate the type of the file that's listed.

|  |  |
| --- | --- |
| **S.No.** | **Prefix & Description** |
| 1 | **-**  Regular file, such as an ASCII text file, binary executable, or hard link. |
| 2 | **B**  Block special file. Block input/output device file such as a physical hard drive. |
| 3 | **C**  Character special file. Raw input/output device file such as a physical hard drive. |
| 4 | **D**  Directory file that contains a listing of other files and directories. |
| 5 | **L**  Symbolic link file. Links on any regular file. |
| 6 | **P**  Named pipe. A mechanism for interprocess communications. |
| 7 | **S**  Socket used for interprocess communication. |

## Metacharacters

Metacharacters have a special meaning in Unix. For example, **\*** and **?** are metacharacters. We use **\*** to match 0 or more characters, a question mark (**?**) matches with a single character.

For Example −

\* :all or any

$ls ch\*.doc

$ls a\*.txt

$ls a\*.\*

$ls \*.txt

Displays all the files, the names of which start with **ch** and end with **.doc** −

ch01-1.doc ch010.doc ch02.doc ch03-2.doc

ch04-1.doc ch040.doc ch05.doc ch06-2.doc

ch01-2.doc ch02-1.doc c

Here, **\*** works as meta character which matches with any character. If you want to display all the files ending with just **.doc**, then you can use the following command −

$ls \*.doc

## Hidden Files

An invisible file is one, the first character of which is the dot or the period character (.). Unix programs (including the shell) use most of these files to store configuration information.

Some common examples of the hidden files include the files −

* **.profile** − The Bourne shell ( sh) initialization script
* **.kshrc** − The Korn shell ( ksh) initialization script
* **.cshrc** − The C shell ( csh) initialization script
* **.rhosts** − The remote shell configuration file

To list the invisible files, specify the **-a** option to **ls** −

$ ls -a

. .profile docs lib test\_results

.. .rhosts hosts pub users

.emacs bin hw1 res.01 work

.exrc ch07 hw2 res.02

.kshrc ch07.bak hw3 res.03

$

* **Single dot (.)** − This represents the current directory.
* **Doub le dot (..)** − This represents the parent directory.

## Creating Files

You can use the **vi** editor to create ordinary files on any Unix system. You simply need to give the following command −

$ vi filename : this file available , then file will be opened

: if file doesn’t’ exist then new file will be created

The above command will open a file with the given filename. Now, press the key **i** to come into the edit mode. Once you are in the edit mode, you can start writing your content in the file as in the following program −

This is unix file....I created it for the first time.....

I'm going to save this content in this file.

Once you are done with the program, follow these steps −

* Press the key **esc** to come out of the edit mode.
* Press two keys **Shift + ZZ** together to come out of the file completely.

You will now have a file created with **filename** in the current directory.

$ vi filename

$

## Editing Files

You can edit an existing file using the **vi** editor. We will discuss in short how to open an existing file −

$ vi filename

~empty row

~empty row

~ empty row

Once the file is opened, you can come in the edit mode by pressing the key **i** and then you can proceed by editing the file. If you want to move here and there inside a file, then first you need to come out of the edit mode by pressing the key **Esc**. After this, you can use the following keys to move inside a file −

* **l** key to move to the right side.
* **h** key to move to the left side.
* **k** key to move upside in the file.
* **j** key to move downside in the file.

So using the above keys, you can position your cursor wherever you want to edit. Once you are positioned, then you can use the **i** key to come in the edit mode. Once you are done with the editing in your file, press **Esc** and finally two keys **Shift + ZZ** together to come out of the file completely.

## Display Content of a File

You can use the **cat** command to see the content of a file. Following is a simple example to see the content of the above created file −

$ cat filename

This is unix file....I created it for the first time.....

I'm going to save this content in this file.

$

You can display the line numbers by using the **-b** option along with the **cat**command as follows −

$ cat -b filename

1 This is unix file....I created it for the first time.....

2 I'm going to save this content in this file.

$

## Counting Words in a File

You can use the **wc** command to get a count of the total number of lines, words, and characters contained in a file. Following is a simple example to see the information about the file created above −

$ wc filename

2 19 103 filename

$

Here is the detail of all the four columns −

* **First Column** − Represents the total number of lines in the file.
* **Second Column** − Represents the total number of words in the file.
* **Third Column** − Represents the total number of bytes in the file. This is the actual size of the file.
* **Fourth Column** − Represents the file name.

You can give multiple files and get information about those files at a time. Following is simple syntax −

$ wc filename1 filename2 filename3

## Copying Files

To make a copy of a file use the **cp** command. The basic syntax of the command is −

$ cp source\_file destination\_file

Following is the example to create a copy of the existing file **filename**.

$ cp filename copyfile

$

You will now find one more file **copyfile** in your current directory. This file will exactly be the same as the original file **filename**.

## Renaming Files

To change the name of a file, use the **mv** command. Following is the basic syntax –

$mv a.txt b/newfile.txt

$mv a.txt b/

$mv a.txt b.txt : rename

$ mv old\_file new\_file

The following program will rename the existing file **filename** to **newfile**.

$ mv filename newfile

$

The **mv** command will move the existing file completely into the new file. In this case, you will find only **newfile** in your current directory.

## Deleting Files

To delete an existing file, use the **rm** command. Following is the basic syntax −

$ rm filename

**Caution** − A file may contain useful information. It is always recommended to be careful while using this **Delete** command. It is better to use the **-i** option along with **rm** command.

Following is the example which shows how to completely remove the existing file **filename**.

$ rm filename

$

You can remove multiple files at a time with the command given below −

$ rm filename1 filename2 filename3

$rmdir : to remove directory

## Standard Unix Streams

Under normal circumstances, every Unix program has three streams (files) opened for it when it starts up −

* **stdin** − This is referred to as the *standard input* and the associated file descriptor is 0. This is also represented as STDIN. The Unix program will read the default input from STDIN.
* **stdout** − This is referred to as the *standard output* and the associated file descriptor is 1. This is also represented as STDOUT. The Unix program will write the default output at STDOUT
* **stderr** − This is referred to as the *standard error* and the associated file descriptor is 2. This is also represented as STDERR. The Unix program will write all the error messages at STDERR.

## Home Directory

The directory in which you find yourself when you first login is called your home directory.

You will be doing much of your work in your home directory and subdirectories that you'll be creating to organize your files.

You can go in your home directory anytime using the following command −

$cd ~

$

Here **~** indicates the home directory. Suppose you have to go in any other user's home directory, use the following command −

$cd ~username

$

To go in your last directory, you can use the following command −

$cd -

$

## Absolute/Relative Pathnames

Directories are arranged in a hierarchy with root (/) at the top. The position of any file within the hierarchy is described by its pathname.

Elements of a pathname are separated by a /. A pathname is absolute, if it is described in relation to root, thus absolute pathnames always begin with a /.

Following are some examples of absolute filenames.

/etc/passwd

/users/sjones/chem/notes

/dev/rdsk/Os3

A pathname can also be relative to your current working directory. Relative pathnames never begin with /. Relative to user amrood's home directory, some pathnames might look like this −

chem/notes

personal/res

To determine where you are within the filesystem hierarchy at any time, enter the command **pwd** to print the current working directory −

$pwd

/user0/home/amrood

$

## Listing Directories

To list the files in a directory, you can use the following syntax −

$ls dirname

Following is the example to list all the files contained in **/usr/local** directory −

$ls /usr/local

X11 bin gimp jikes sbin

ace doc include lib share

atalk etc info man ami

## Creating Directories

We will now understand how to create directories. Directories are created by the following command −

$mkdir dirname

Here, directory is the absolute or relative pathname of the directory you want to create. For example, the command −

$mkdir mydir

$

Creates the directory **mydir** in the current directory. Here is another example −

$mkdir /tmp/test-dir

$

This command creates the directory **test-dir** in the **/tmp** directory. The **mkdir** command produces no output if it successfully creates the requested directory.

If you give more than one directory on the command line, **mkdir** creates each of the directories. For example, −

$mkdir docs pub

$

Creates the directories docs and pub under the current directory.

## Creating Parent Directories

We will now understand how to create parent directories. Sometimes when you want to create a directory, its parent directory or directories might not exist. In this case, **mkdir** issues an error message as follows −

$mkdir /tmp/amrood/test

mkdir: Failed to make directory "/tmp/amrood/test";

No such file or directory

$

In such cases, you can specify the **-p** option to the **mkdir** command. It creates all the necessary directories for you. For example −

$mkdir -p /tmp/amrood/test

$

The above command creates all the required parent directories.

## Removing Directories

Directories can be deleted using the **rmdir** command as follows −

$rmdir dirname

$

**Note** − To remove a directory, make sure it is empty which means there should not be any file or sub-directory inside this directory.

You can remove multiple directories at a time as follows −

$rmdir dirname1 dirname2 dirname3

$

The above command removes the directories dirname1, dirname2, and dirname3, if they are empty. The **rmdir** command produces no output if it is successful.

## Changing Directories

You can use the **cd** command to do more than just change to a home directory. You can use it to change to any directory by specifying a valid absolute or relative path. The syntax is as given below −

$cd dirname

$

Here, **dirname** is the name of the directory that you want to change to. For example, the command −

$cd /usr/local/bin

$

Changes to the directory **/usr/local/bin**. From this directory, you can **cd** to the directory **/usr/home/amrood** using the following relative path −

$cd ../../home/amrood

$

## Renaming Directories

The **mv (move)** command can also be used to rename a directory. The syntax is as follows −

$mv olddir newdir

$

You can rename a directory **mydir** to **yourdir** as follows −

$mv mydir yourdir

$

## The directories . (dot) and .. (dot dot)

The **filename .** (dot) represents the current working directory; and the **filename ..** (dot dot) represents the directory one level above the current working directory, often referred to as the parent directory.

If we enter the command to show a listing of the current working directories/files and use the **-a option** to list all the files and the **-l option** to provide the long listing, we will receive the following result.

$ls -la

drwxrwxr-x 4 teacher class 2048 Jul 16 17.56 .

drwxr-xr-x 60 root 1536 Jul 13 14:18 ..

---------- 1 teacher class 4210 May 1 08:27 .profile

-rwxr-xr-x 1 teacher class 1948 May 12 13:42 memo

$

# Linux - File Permission / Access Modes

* **Owner permissions/user permission**  − The owner's permissions determine what actions the owner of the file can perform on the file.
* **Group permissions** − The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
* **Other (world) permissions** − The permissions for others indicate what action all other users can perform on the file.

## The Permission Indicators

While using **ls -l** command, it displays various information related to file permission as follows −

$ls -l /home/amrood

-rwxr-xr-- 1 amrood users 1024 Nov 2 00:10 myfile

drwxr-xr--- 1 amrood users 1024 Nov 2 00:10 mydir

Here, the first column represents different access modes, i.e., the permission associated with a file or a directory.

The permissions are broken into groups of threes, and each position in the group denotes a specific permission, in this order: read (r), write (w), execute (x) −

* The first three characters (2-4) represent the permissions for the file's owner. For example, **-rwxr-xr--** represents that the owner has read (r), write (w) and execute (x) permission.
* The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example, **-rwxr-xr--**represents that the group has read (r) and execute (x) permission, but no write permission.
* The last group of three characters (8-10) represents the permissions for everyone else. For example, **-rwxr-xr--** represents that there is **read (r)** only permission.

## File Access Modes

The permissions of a file are the first line of defense in the security of a Unix system. The basic building blocks of Unix permissions are the **read**, **write**, and **execute** permissions, which have been described below −

### Read

Grants the capability to read, i.e., view the contents of the file.

### Write

Grants the capability to modify, or remove the content of the file.

### Execute

User with execute permissions can run a file as a program.

## Directory Access Modes

Directory access modes are listed and organized in the same manner as any other file. There are a few differences that need to be mentioned −

### Read

Access to a directory means that the user can read the contents. The user can look at the **filenames** inside the directory.

### Write

Access means that the user can add or delete files from the directory.

### Execute

Executing a directory doesn't really make sense, so think of this as a traverse permission.

A user must have **execute** access to the **bin** directory in order to execute the **ls** or the **cd** command.

## Changing Permissions

To change the file or the directory permissions, you use the **chmod** (change mode) command. There are two ways to use chmod — the symbolic mode and the absolute mode.

### Using chmod in Symbolic Mode

The easiest way for a beginner to modify file or directory permissions is to use the symbolic mode. With symbolic permissions you can add, delete, or specify the permission set you want by using the operators in the following table.

|  |  |
| --- | --- |
| **S.No.** | **Chmod operator & Description** |
| 1 | **+**  Adds the designated permission(s) to a file or directory. |
| 2 | **-**  Removes the designated permission(s) from a file or directory. |
| 3 | **=**  Sets the designated permission(s). |

Here's an example using **testfile**. Running **ls -1** on the testfile shows that the file's permissions are as follows −

$ls -l testfile

-rwxrwxr-- 1 amrood users 1024 Nov 2 00:10 testfile

Then each example **chmod** command from the preceding table is run on the testfile, followed by **ls –l**, so you can see the permission changes –

U – use r

G – group

O – other

$chmod o+wx testfile/dirname

$ls -l testfile

-rwxrwxrwx 1 amrood users 1024 Nov 2 00:10 testfile

$chmod o-x testfile/dirname

$ls -l testfile

-rwxrwxrw- 1 amrood users 1024 Nov 2 00:10 testfile

$chmod u-x testfile

$ls -l testfile

-rw-rwxrwx 1 amrood users 1024 Nov 2 00:10 testfile

$chmod g = rx testfile

$ls -l testfile

-rw-r-xrwx 1 amrood users 1024 Nov 2 00:10 testfile

Here's how you can combine these commands on a single line −

$chmod o+wx,u-x,g = rx testfile

$ls -l testfile

-rw-r-xrwx 1 amrood users 1024 Nov 2 00:10 testfile

## Using chmod with Absolute Permissions

The second way to modify permissions with the chmod command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set.

|  |  |  |
| --- | --- | --- |
| **Number** | **Octal Permission Representation** | **Ref** |
| **0** | No permission | --- |
| **1** | Execute permission | --x |
| **2** | Write permission | -w- |
| **3** | Execute and write permission: 1 (execute) + 2 (write) = 3 | -wx |
| **4** | Read permission | r-- |
| **5** | Read and execute permission: 4 (read) + 1 (execute) = 5 | r-x |
| **6** | Read and write permission: 4 (read) + 2 (write) = 6 | rw- |
| **7** | All permissions: 4 (read) + 2 (write) + 1 (execute) = 7 | rwx |

Here's an example using the testfile. Running **ls -1** on the testfile shows that the file's permissions are as follows −

$ls -l testfile

-rwxrwxr-- 1 amrood users 1024 Nov 2 00:10 testfile

Then each example **chmod** command from the preceding table is run on the testfile, followed by **ls –l**, so you can see the permission changes −

$ chmod 755 testfile

$ls -l testfile

-rwxr-xr-x 1 amrood users 1024 Nov 2 00:10 testfile

$chmod 743 testfile

$ls -l testfile

-rwxr---wx 1 amrood users 1024 Nov 2 00:10 testfile

$chmod 043 testfile

$ls -l testfile

----r---wx 1 amrood users 1024 Nov 2 00:10 testfile

## Changing Owners and Groups

While creating an account on Unix, it assigns a **owner ID** and a **group ID** to each user. All the permissions mentioned above are also assigned based on the Owner and the Groups.

Two commands are available to change the owner and the group of files −

* **chown** − The **chown** command stands for **"change owner"** and is used to change the owner of a file.
* **chgrp** − The **chgrp** command stands for **"change group"** and is used to change the group of a file.

## Changing Ownership

The **chown** command changes the ownership of a file. The basic syntax is as follows −

$ chown user filelist

The value of the user can be either the **name of a user** on the system or the **user id (uid)** of a user on the system.

The following example will help you understand the concept −

$ chown amrood testfile

$

Changes the owner of the given file to the user **amrood**.

**NOTE** − The super user, root, has the unrestricted capability to change the ownership of any file but normal users can change the ownership of only those files that they own.

## Changing Group Ownership

The **chgrp** command changes the group ownership of a file. The basic syntax is as follows −

$ chgrp group filelist

The value of group can be the **name of a group** on the system or **the group ID (GID)** of a group on the system.

Following example helps you understand the concept −

$ chgrp special testfile

$

Changes the group of the given file to **special** group.

## SUID and SGID File Permission

Often when a command is executed, it will have to be executed with special privileges in order to accomplish its task.

As an example, when you change your password with the **passwd** command, your new password is stored in the file **/etc/shadow**.

As a regular user, you do not have **read** or **write** access to this file for security reasons, but when you change your password, you need to have the write permission to this file. This means that the **passwd** program has to give you additional permissions so that you can write to the file **/etc/shadow**.

Additional permissions are given to programs via a mechanism known as the **Set User ID (SUID)** and **Set Group ID (SGID)** bits.

When you execute a program that has the SUID bit enabled, you inherit the permissions of that program's owner. Programs that do not have the SUID bit set are run with the permissions of the user who started the program.

This is the case with SGID as well. Normally, programs execute with your group permissions, but instead your group will be changed just for this program to the group owner of the program.

The SUID and SGID bits will appear as the letter **"s"** if the permission is available. The SUID **"s"** bit will be located in the permission bits where the owners’ **execute** permission normally resides.

For example, the command −

$ ls -l /usr/bin/passwd

-r-sr-xr-x 1 root bin 19031 Feb 7 13:47 /usr/bin/passwd\*

$

Shows that the SUID bit is set and that the command is owned by the root. A capital letter **S** in the execute position instead of a lowercase **s** indicates that the execute bit is not set.

If the sticky bit is enabled on the directory, files can only be removed if you are one of the following users −

* The owner of the sticky directory
* The owner of the file being removed
* The super user, root

To set the SUID and SGID bits for any directory try the following command −

$ chmod ug+s dirname

$ ls -l

drwsr-sr-x 2 root root 4096 Jun 19 06:45 dirname

$

# Linux - Pipes and Filters

To make a pipe, put a vertical bar (**|**) on the command line between two commands.

When a program takes its input from another program, it performs some operation on that input, and writes the result to the standard output. It is referred to as a ***filter***.

## The grep Command

The grep command searches a file or files for lines that have a certain pattern. The syntax is −

$grep pattern file(s)

The name **"grep"** comes from the ed (a Unix line editor) command **g/re/p**which means “globally search for a regular expression and print all lines containing it”.

A regular expression is either some plain text (a word, for example) and/or special characters used for pattern matching.

The simplest use of grep is to look for a pattern consisting of a single word. It can be used in a pipe so that only those lines of the input files containing a given string are sent to the standard output. If you don't give grep a filename to read, it reads its standard input; that's the way all filter programs work −

$ls -l | grep "Aug"

-rw-rw-rw- 1 john doc 11008 Aug 6 14:10 ch02

-rw-rw-rw- 1 john doc 8515 Aug 6 15:30 ch07

-rw-rw-r-- 1 john doc 2488 Aug 15 10:51 intro

-rw-rw-r-- 1 carol doc 1605 Aug 23 07:35 macros

$

There are various options which you can use along with the **grep** command −

|  |  |
| --- | --- |
| **S.No.** | **Option & Description** |
| 1 | **-v**  Prints all lines that do not match pattern. |
| 2 | **-n**  Prints the matched line and its line number. |
| 3 | **-l**  Prints only the names of files with matching lines (letter "l") |
| 4 | **-c**  Prints only the count of matching lines. |
| 5 | **-I**  Matches either upper or lowercase. |

Let us now use a regular expression that tells grep to find lines with **"carol"**, followed by zero or other characters abbreviated in a regular expression as ".\*"), then followed by "Aug".−

Here, we are using the ***-i*** option to have case insensitive search −

$ls -l | grep -i "carol.\*aug"

-rw-rw-r-- 1 carol doc 1605 Aug 23 07:35 macros

$

## The sort Command

The **sort** command arranges lines of text alphabetically or numerically. The following example sorts the lines in the food file −

$sort filename.txt

Afghani Cuisine

Bangkok Wok

Big Apple Deli

Isle of Java

Mandalay

Sushi and Sashimi

Sweet Tooth

Tio Pepe's Peppers

$

The **sort** command arranges lines of text alphabetically by default. There are many options that control the sorting −

|  |  |
| --- | --- |
| **S.No.** | **Description** |
| 1 | **-n**  Sorts numerically (example: 10 will sort after 2), ignores blanks and tabs. |
| 2 | **-r**  Reverses the order of sort. |
| 3 | **-f**  Sorts upper and lowercase together. |
| 4 | **+x**  Ignores first **x** fields when sorting. |

More than two commands may be linked up into a pipe. Taking a previous pipe example using **grep**, we can further sort the files modified in August by the order of size.

The following pipe consists of the commands **ls**, **grep**, and **sort** −

$ls -l | grep "Aug" | sort +4n

-rw-rw-r-- 1 carol doc 1605 Aug 23 07:35 macros

-rw-rw-r-- 1 john doc 2488 Aug 15 10:51 intro

-rw-rw-rw- 1 john doc 8515 Aug 6 15:30 ch07

-rw-rw-rw- 1 john doc 11008 Aug 6 14:10 ch02

$

This pipe sorts all files in your directory modified in August by the order of size, and prints them on the terminal screen. The sort option +4n skips four fields (fields are separated by blanks) then sorts the lines in numeric order.

## The pg and more Commands

A long output can normally be zipped by you on the screen, but if you run text through more or use the **pg** command as a filter; the display stops once the screen is full of text.

Let's assume that you have a long directory listing. To make it easier to read the sorted listing, pipe the output through **more** as follows −

$ls -l | grep "Aug" | sort +4n | more

-rw-rw-r-- 1 carol doc 1605 Aug 23 07:35 macros

-rw-rw-r-- 1 john doc 2488 Aug 15 10:51 intro

-rw-rw-rw- 1 john doc 8515 Aug 6 15:30 ch07

-rw-rw-r-- 1 john doc 14827 Aug 9 12:40 ch03

.

.

.

-rw-rw-rw- 1 john doc 16867 Aug 6 15:56 ch05

--More--(74%)

The screen will fill up once the screen is full of text consisting of lines sorted by the order of the file size. At the bottom of the screen is the **more** prompt, where you can type a command to move through the sorted text.

Once you're done with this screen, you can use any of the commands listed in the discussion of the more program.