5.1 Introduction

Since everything is considered a file in Linux, file management is an important topic. Not only are your documents considered files, but hardware devices such as hard drives and memory as well. Even directories are considered files, since they are special files that are used to contain other files.

The essential commands needed for file management are often named by short abbreviations of their functions. You can use the ls command to list files, the cpcommand to copy files, the mv command to move or rename files and the rmcommand to delete files.

For working with directories, use the mkdir command to make directories, the rmdir command to remove directories (if they are empty) and the rm command to recursively delete directories containing files.

5.2 ls Command

While technically not a file management command, the ability to list files using lsis critical to file management. By default, the ls command will list the files in the current directory:

```
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos  test
```

For arguments, the ls command will accept an arbitrary number of different pathnames to attempt to list:

```
sysadmin@localhost:~$ ls . test /usr/local
.:
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos  test

/usr/local:
bin  etc  games  include  lib  man  sbin  share  src

test:
adjectives.txt          alpha.txt    linux.txt     people.csv
alpha-first.txt         animals.txt  longfile.txt  profile.txt
alpha-first.txt.original  food.txt     newhome.txt   red.txt
alpha-second.txt        hidden.txt   numbers.txt
alpha-third.txt         letters.txt  os.csv
```

Note the use of the . character. When used where a directory is expected, it represents the current directory. However, when the . character is used at the beginning of a file or directory name, it makes the file hidden by default. These *hidden files* are not normally displayed when using the ls command, though many can be commonly found in user home directories. Hidden files and directories are typically used for individual specific data or settings. Using the -aoption will display all files, including hidden files:

```
sysadmin@localhost:~$ ls -a
.           .bashrc   .selected_editor  Downloads  Public     test
..          .cache    Desktop          Music      Templates
.bash_logout  .profile  Documents        Pictures   Videos
```

The -A option to the ls command will display almost all files of a directory. The current directory file . and the parent directory file .. are omitted.

```
sysadmin@localhost:~$ ls -A
.bash_logout  .profile       Documents  Pictures  Videos
.bashrc       .selected_editor  Downloads  Public    test
.cache        Desktop        Music      Templates
```

To learn the details about a file, such as the type of file, the permissions, ownerships or the timestamp, perform a long listing. To get a detailed view, use the -l option to the ls command. In the example below, the /var/logdirectory is being listed because of its variety of output:

```
sysadmin@localhost:~$ ls -l /var/log/
total 1812
-rw-r--r-- 1          root        root        14100 Jul 23 18:45 alternatives.log
drwxr-xr-x 1          root        root        38 Jul 23 18:43 apt
-rw-r----- 1          syslog      adm         23621 Aug 23 15:17 auth.log
-rw-r--r-- 1          root        root        47816 Jul 17 03:36 bootstrap.log
-rw-rw---- 1          root        utmp        0 Jul 17 03:34 btmp
-rw-r----- 1          syslog      adm         8013 Aug 23 15:17 cron.log
-rw-r----- 1          root        adm         85083 Jul 23 18:45 dmesg
-rw-r--r-- 1          root        root        245540 Jul 23 18:45 dpkg.log
-rw-r--r-- 1          root        root        32064 Jul 23 18:45 faillog
drwxr-xr-x 1          root        root        32 Jul 17 03:36 fsck
-rw-r----- 1          syslog      adm         416 Aug 22 15:43 kern.log
-rw-rw-r-- 1          root        utmp        292584 Aug 20 18:44 lastlog
-rw-r----- 1          syslog      adm         0 Aug 22 06:25 syslog
-rw-r----- 1          syslog      adm         1087150 Aug 23 15:17 syslog.1
drwxr-xr-x 1          root        root        0 Apr 11 21:58 upstart
-rw-rw-r-- 1          root        utmp        384 Aug 20 18:44 wtmp
```

Viewing the above output as fields that are separated by spaces, they indicate:

- **File Type**

  - - rw-r--r-- 1 root root 14100 Jul 23 18:45 alternatives.log
  - d rwxr-xr-x 1 root root 38 Jul 23 18:43 apt

  The first field actually contains eleven characters, where the first character indicates the type of file and the next ten specify permissions. The file types are:

  - d for directory
  - - for regular file
  - l for a symbolic link
  - s for a socket
  - p for a pipe
  - b for a block file
  - c for a character file

Note that alternatives.log is a regular file -, while the apt is a directory d.

Permissions

```
d rwxr-xr-x 1 root root              0 Apr 11 21:58 upstart
```

The permissions are read r, write w, and execute x. Breaking this down a bit:

- rwx : Owner Permissions
- r-x : Group Permissions
- r-x : Everyone Else's Permissions

- **Hard Link Count**

```
-rw-r----- 1 syslog adm 23621 Aug 23 15:17 auth.log
```

There is only one directory name that links to this file. Hard links will be discussed in detail later in the course.

- **User Owner**

```
-rw-r----- 1 syslog adm 416 Aug 22 15:43 kern.log
```

User syslog owns this file. Every time a file is created, the ownership is automatically assigned to the user who created it.

- **Group Owner**

```
-rw-rw-r-- 1 root utmp 292584 Aug 20 18:44 lastlog
```

Although root created this file, any member of the utmp group has read and write access to it (as indicated by the group permissions).

- **File Size**

```
-rw-r----- 1 syslog adm 1087150 Aug 23 15:17 syslog.1
```

The size of the file in bytes. In the case of a directory, it might actually be a multiple of the block size used for the file system.

- **Timestamp**

```
drwxr-xr-x 1 root root 32 Jul 17 03:36 fsck
```

This indicates the time that the file's contents were last modified. This time will be listed as just the date and year if the file was last modified more than six months from the current date. Otherwise, the month, day, and time is displayed. Using the ls command with the --full-time option will display timestamps in full detail.

- **Filename**

```
-rw-r--r-- 1 root root 47816 Jul 17 03:36 bootstrap.log
```

The final field contains the name of the file or directory. In the case of symbolic links, the link name will be shown along with an arrow and the pathname of the file that is linked is shown.

```
lrwxrwxrwx. 1 root root 22 Nov 6 2012 /etc/grub.conf -> ../boot/grub/grub.conf
```

Sorting

The output of the ls command is sorted alphabetically by file name. It can sort by other methods as well:

- `-S` : Sort by files by size
- `-t` : Sort by timestamp
- `-r` : Reverses any type of sort

With both time and size sorts, add the `-l` option or the `--full-time` option, to be able to view those details:

```
sysadmin@localhost:~$ ls -t --full-time
total 0
drwxr-xr-x 1 sysadmin sysadmin   0 2014-09-18 22:25:18.000000000 +0000 Desktop
drwxr-xr-x 1 sysadmin sysadmin   0 2014-09-18 22:25:18.000000000 +0000 Documents
drwxr-xr-x 1 sysadmin sysadmin   0 2014-09-18 22:25:18.000000000 +0000 Downloads
drwxr-xr-x 1 sysadmin sysadmin   0 2014-09-18 22:25:18.000000000 +0000 Music
drwxr-xr-x 1 sysadmin sysadmin   0 2014-09-18 22:25:18.000000000 +0000 Pictures
drwxr-xr-x 1 sysadmin sysadmin   0 2014-09-18 22:25:18.000000000 +0000 Public
drwxr-xr-x 1 sysadmin sysadmin   0 2014-09-18 22:25:18.000000000 +0000 Templates
drwxr-xr-x 1 sysadmin sysadmin   0 2014-09-18 22:25:18.000000000 +0000 Videos
drwxr-xr-x 1 sysadmin sysadmin 420 2014-09-18 22:25:18.000000000 +0000 test
```

Recursion

When managing files it is important to understand what the term *recursive option* means. When the recursive option is used with file management commands, it means to apply that command to not only the specified directory, but also to all subdirectories and all of the files within all subdirectories.

Some commands use `-r` for recursive while others use `-R`. The `-R` option is used for recursive with the `ls` command because the `-r` option is used for reversing how the files are sorted:

```
sysadmin@localhost:~$ ls -lR /var/log
/var/log:
total 804
-rw-r--r-- 1 root   root  14100 Sep 18 22:25 alternatives.log
drwxr-xr-x 1 root   root     38 Sep 18 22:23 apt
-rw-r----- 1 syslog adm    1343 Sep 29 21:17 auth.log
-rw-r--r-- 1 root   root  47816 Sep 17 03:35 bootstrap.log
-rw-rw---- 1 root   utmp      0 Sep 17 03:34 btmp
-rw-r----- 1 syslog adm     546 Sep 29 21:17 cron.log
-rw-r----- 1 root   adm   85083 Sep 18 22:25 dmesg
-rw-r--r-- 1 root   root 252496 Sep 18 22:25 dpkg.log
-rw-r--r-- 1 root   root  32064 Sep 18 22:25 faillog
drwxr-xr-x 1 root   root     32 Sep 17 03:35 fsck
-rw-r----- 1 syslog adm     108 Sep 29 18:48 kern.log
-rw-rw-r-- 1 root   utmp 292584 Sep 29 18:48 lastlog
-rw-r----- 1 syslog adm   58355 Sep 29 21:17 syslog
drwxr-xr-x 1 root   root      0 Apr 11 21:58 upstart
-rw-rw-r-- 1 root   utmp    384 Sep 29 18:48 wtmp
```

```
/var/log/apt:
total 20
-rw-r--r-- 1 root root 13672 Sep 18 22:25 history.log
-rw-r----- 1 root adm    402 Sep 18 22:25 term.log

/var/log/fsck:
total 8
-rw-r----- 1 root adm 31 Sep 17 03:35 checkfs
-rw-r----- 1 root adm 31 Sep 17 03:35 checkroot

/var/log/upstart:
total 0
```

Use the recursive option carefully because its impact can be huge! With the `ls`command, using the `-R` option can result in a large amount of output in your terminal. For the other file management commands, the recursive option can impact a large amount of files and disk space.

The `-d` option to the `ls` command also important. When listing a directory, the `ls` command normally will show the contents of that directory, but when the `-d`option is added, then it will display the directory itself:

```
sysadmin@localhost:~$ ls -ld /var/log
drwxrwxr-x 1 root syslog 216 Sep 29 18:48 /var/log
```

5.3 file Command

The `file` command "looks at" the contents of a file to report what kind of file it is; it does this by matching the content to known types stored in a "magic" file. Many commands that you will use in Linux expect that the file name provided as an argument is a text file (rather than some sort of binary file). As a result, it is a good idea to use the `file` command before attempting to access a file to make sure that it does contain text. For example:

```
sysadmin@localhost:~$ file test/newhome.txt
test/newhome.txt: ASCII text
sysadmin@localhost:~$ file /bin/ls
/bin/ls: ELF 64-bit LSB  executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for G
NU/Linux 2.6.24, BuildID[sha1]=64d095bc6589dd4bfbf1c6d62ae985385965461b, stripped
```

Consider This

If you use a command that is expecting a text file argument, you may get strange results if you provide a non-text file. Your terminal may become disrupted by attempting to output binary content; as a result, the terminal may not be able to function properly any more.

The result will be "weird looking" characters being displayed in the terminal. To correct this situation, typing the command `reset` will often work. If this does not solve the problem, closing the terminal and opening a new one will solve the problem.

5.4 touch Command

The `touch` command performs two functions:

- create an empty file
- update the modification timestamp on an existing file

If the argument provided is an existing file, then the `touch` command will update the file's timestamp:

```
sysadmin@localhost:~$ ls -l test/red.txt
-rw-r--r-- 1 sysadmin sysadmin 51 Sep 18 22:25 test/red.txt
sysadmin@localhost:~$ touch test/red.txt
sysadmin@localhost:~$ ls -l test/red.txt
-rw-r--r-- 1 sysadmin sysadmin 51 Sep 29 22:35 test/red.txt
```

If the argument is a file that does not exist, a new file is created with the current time for the timestamp. The contents of this new file will be empty:

```
sysadmin@localhost:~$ touch newfile
sysadmin@localhost:~$ ls
Desktop  Downloads  Pictures  Templates  newfile
Documents  Music    Public   Videos    test
sysadmin@localhost:~$ ls -l newfile
-rw-rw-r-- 1 sysadmin sysadmin 0 Sep 29 22:39 newfile
```

Each file has three timestamps:

- The last time the file's contents were modified. This is the timestamp provided by the `ls -l` command by default. The `touch` command modified this timestamp by default.
- The last time the file was accessed. To modify this timestamp, use the `touch -a` command.
- The last time the file attributes, like permissions or ownership, were changed. These file attributes are also called the file's *metadata*. To modify this timestamp, use the `touch -c` command.

The `touch` command will normally update the specified time to the current time, but `-t` option with a timestamp value to can be used instead.

```
sysadmin@localhost:~$ touch -t 201412251200 newfile
sysadmin@localhost:~$ ls -l newfile
-rw-rw-r-- 1 sysadmin sysadmin 0 Dec 25  2014 newfile
```

In order to view all three timestamps that are kept for a file, use the `stat`command with the path to the file as an argument:

```
sysadmin@localhost:~$ stat test/alpha.txt
  File: 'test/alpha.txt'
  Size: 390           Blocks: 8          IO Block: 4096   regular file
Device: 58h/88d Inode: 13987       Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1001/sysadmin)   Gid: ( 1001/sysadmin)
Access: 2014-09-23 13:52:03.070012387 +0000
Modify: 2014-09-18 22:25:18.000000000 +0000
Change: 2014-09-23 13:52:03.070012387 +0000
```

Birth: -

5.5 cp Command

Creating copies of files can be useful for numerous reasons:

- If a copy of a file is created before changes are made, then it is possible to revert back to the original.
- It can be used to transfer a file to removable media devices.
- A copy of an existing document can be made, so to take advantage of the existing layout and content to get started more quickly than from scratch.

Consider This

While permissions will be covered in greater detail later in the course, they can have an impact on file management commands, such as the cpcommand.

For example, in order to be able to copy a file, you will need to have execute permission to access the directory where the file is located and the read permission for the file you are trying to copy.

You will also need to have write and execute permission on the directory where you want to put the copied file. Typically, there are two places where you should always have write and execute permission on the directory: your home directory (for "permanent" files) and the /tmp directory (for "temporary" files).

The copy command cp takes at least two arguments: the first argument is the path to the file to be copied and the second argument is the path to where the copy will be placed. The files to be copied are sometimes referred to as the source and the place to where the copies are placed in is called the destination.

Recall that the ~ character that represents your home directory; it is commonly used as a source or destination.

```
sysadmin@localhost:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos  test
```

For example, to copy the /etc/services file to your home directory, you could use the following command:

```
sysadmin@localhost:~$ cp /etc/services ~
```

The result of executing the previous command would create a copy of the contents of the /etc/services file in your home directory; the new file in your home directory would also be named services.

```
sysadmin@localhost:~$ ls
Desktop    Downloads  Pictures  Templates  services
Documents  Music      Public    Videos     test
```

To copy a file and rename the file in one step you can execute a command like:

```
sysadmin@localhost:~$ cp /etc/services ~/ports
```

The previous command would copy the contents of the /etc/services file into a new file named ports in the home directory.

To copy multiple files into a directory, additional file names to copy can be included as long as the last argument is a destination directory. Although they may only represent one argument, a wildcard pattern can be expanded to match multiple pathnames. These will be covered in detail in the next

chapter. The following command copies all files in the test directory that start with an n to the home directory, as a result newhome.txt and numbers.txt are copied.

```
sysadmin@localhost:~$ cp test/n* ~
sysadmin@localhost:~$ ls
Desktop   Downloads  Pictures  Templates  newhome.txt  ports    test
Documents  Music     Public    Videos     numbers.txt  services
```

An issue that frequently comes up when using wildcard patterns with the cpcommand is that sometimes they match the names of directories as well as regular files. If an attempt is made to copy a directory, then the cp command will print an error message. Using the -v option makes the cp command print verbose message, so you can always tell what copies succeed as well as those that fail. In the following wildcard example we've marked in red the lines which indicate that the directories were matched, but not copied.

```
sysadmin@localhost:~$ cp -v /etc/b* ~
'/etc/bash.bashrc' -> '/home/sysadmin/bash.bashrc'
cp: omitting directory '/etc/bash_completion.d'
cp: omitting directory '/etc/bind'
'/etc/bindresvport.blacklist' -> '/home/sysadmin/bindresvport.blacklist'
'/etc/blkid.conf' -> '/home/sysadmin/blkid.conf'
```

In order to copy a directory, its contents must be copied as well, including all files within the directories and all of its subdirectories. This can be done by using a recursive option of either -r or -R.

```
sysadmin@localhost:~$ cp -R -v /etc/perl ~
'/etc/perl' -> '/home/sysadmin/perl'
'/etc/perl/CPAN' -> '/home/sysadmin/perl/CPAN'
'/etc/perl/Net' -> '/home/sysadmin/perl/Net'
'/etc/perl/Net/libnet.cfg' -> '/home/sysadmin/perl/Net/libnet.cfg'
```

Consider This

The archive option -a copies the contents of the file and also attempts to maintain the original timestamps and file ownership. For regular users, only original timestamps can be maintained, since regular users can't create files owned by other users. If the root user uses the -a option, then the cpcommand will create a new file owned by the original file owner and also use the original file's timestamps.

The -a option also implies that recursion will be done, so it can also be used to copy a directory.

5.6 mv Command

The mv command is used to move a file from one pathname in the filesystem to another. When you move a file, it's like creating a copy of the original file with a new pathname and then deleting the original file.

Consider This

While permissions will be covered in greater detail later in the course, they can have an impact on file management commands, such as the mvcommand. Moving a file requires write and execute permissions on both the origin and destination directories.

If you move a file from one directory to another and you don't specify a new name for the file, then it will retain its original name. For example, the following will move the ~/test/red.txt file into the home directory ("~" refers to the user's home directory) and the resulting file name will be red.txt:

```
sysadmin@localhost:~/test$ mv red.txt ~
sysadmin@localhost:~/test$ ls ~
Desktop  Downloads  Pictures  Templates  red.txt
Documents  Music    Public   Videos    test
```

Like the cp command, the mv command will allow you to specify multiple files to move, as long as the final argument provided to the command is a directory. For example:

```
sysadmin@localhost:~/test$ mv animals.txt food.txt alpha.txt /tmp
sysadmin@localhost:~/test$ ls /tmp
alpha.txt  animals.txt  food.txt
```

There is no need for a recursive option with the mv command. When a directory is moved, everything it contains is automatically moved as well.

Moving a file within the same directory is an effective way to rename it. For example, in the following command the people.csv file is moved from the current directory to the current directory and given a new name of founders.csv. In other words people.csv is renamed founders.csv:

```
sysadmin@localhost:~/test$ mv people.csv founders.csv
```

5.7 rm Command

The rm command is used to delete files and directories. It is important to keep in mind that deleted files and directories do not go into a "trash can" as with desktop oriented operating systems. When a file is deleted with the rm command, it is permanently gone.

Without any options, the rm command is typically used to remove regular files:

```
sysadmin@localhost:~test$ rm alpha.txt
sysadmin@localhost:~test$ ls alpha.txt
ls: cannot access alpha.txt: No such file or directory
```

Extra care should be applied when using wildcards to specify which files to remove, as the extent to which the pattern might match files may be beyond what was anticipated. To avoid accidently deleting files when using globbing characters, use the -i option. This option makes the rm command confirm "interactively" every file that you delete:

```
sysadmin@localhost:~test$ rm -i a*
rm: remove regular file 'adjectives.txt'? y
rm: remove regular file 'alpha-first.txt'? y
rm: remove regular file 'alpha-first.txt.original'? y
rm: remove regular file 'alpha-second.txt'? y
rm: remove regular file 'alpha-third.txt'? y
rm: remove regular file 'animals.txt'? y
```

Some distributions make the -i option a default option by making an alias for the rm command.

The rm command will ignore directories that it's asked to remove; to delete a directory, use either the -r or -R options. Just be careful as this will delete all files and all subdirectories:

```
sysadmin@localhost:~$ rm test
```

```
rm: cannot remove 'test': Is a directory
sysadmin@localhost:~$ rm -r test
```

Consider This

While permissions will be covered in greater detail later in the course, they can have an impact on file management commands, such as the `rm`command.

To delete a file within a directory, a user must have write and execute permission on a directory. Regular users typically only have this type of permission in their home directory and its subdirectories.

The /tmp and /var/tmp directories do have the special permission called *sticky bit* set on them, so that files in these directories can only be deleted by the user that owns them (with the exception of the root user who can delete any file in any directory). So this means if you copy a file to the /tmpdirectory, then other users of the system will not be able to delete your file.

5.8 mkdir Command

The `mkdir` command allows you to create a directory. Creating directories is an essential file management skill, since you will want to maintain some functional organization with your files and not have them all placed in a single directory.

Typically, you only have a handful of directories in your home directory by default. Exactly what directories you have will vary due to the distribution of Linux, what software has been installed on your system and actions that may have been taken by the administrator.

For example, upon successful graphical login on a default installation of CentOS, the following directories have already been created automatically in the user's home directory: Desktop, Documents, Downloads, Music, Pictures, Public, Templates and Videos.

The bin directory is a common directory for users to create in their home directory. It is a useful directory to place *scripts* that the user has created. In the following example, the user creates a bin directory within their home directory:

```
sysadmin@localhost:~$mkdir bin
sysadmin@localhost:~$ ls
Desktop  Downloads  Pictures  Templates  bin
Documents  Music  Public  Videos  test
```

The `mkdir` command can accept a list of space separated pathnames for new directories to create:

```
sysadmin@localhost:~$mkdir one two three
sysadmin@localhost:~$ ls
Desktop  Downloads  Pictures  Templates  bin  test  two
Documents  Music  Public  Videos  one  three
```

Consider This

While permissions will be covered in greater detail later in the course, they can have an impact on file management commands, such as the `mkdir`command.

Creating a directory requires write and execute permissions for the parent of the proposed directory. For example, to issue the following command, it is necessary to have write and execute permission in the /etc directory: `mkdir /etc/test`

Directories are often described in terms of their relationship to each other. If one directory contains another directory, then it is referred to as the parent directory. The subdirectory is referred to as a child directory. In the following example /home is the parent directory and /sysadmin is the child directory:

```
sysadmin@localhost:~$ pwd
/home/sysadmin
```

When creating a child directory, its parent directory must first exist. If an administrator wanted to create directory /blue inside of /home/sysadmin/redwithout its existence, there would be an error:

```
sysadmin@localhost:~$ mkdir /home/sysadmin/red/blue
mkdir: cannot create directory '/home/sysadmin/red/blue': No such file or directory
```

By adding the `-p` option, the mkdir command automatically creates the parent directories for any child directories about to be created. This is especially useful for making "deep" pathnames:

```
sysadmin@localhost:~$ mkdir -p /home/sysadmin/red/blue/yellow/green
sysadmin@localhost:~$ ls -R red
red:
blue

red/blue:
yellow

red/blue/yellow:
green

red/blue/yellow/green:
```

5.9 rmdir Command

The `rmdir` command is used to remove *empty* directories.

```
sysadmin@localhost:~$ rmdir bin
```

Using the `-p` option with the `rmdir` command will remove directory paths, but only if all of the directories contain other empty directories.

```
sysadmin@localhost:~$ rmdir red
rmdir: failed to remove 'red': Directory not empty
sysadmin@localhost:~$ rmdir -p red/blue/yellow/green
```

Otherwise, if a directory contains anything except other directories, you'll need to use the command `rm` with a recursive option:

```
sysadmin@localhost:~$ rmdir test
rmdir: failed to remove 'test': Directory not empty
sysadmin@localhost:~$ rm -r test
```

Consider This

While permissions will be covered in greater detail later in the course, they can have an impact on file management commands, such as the `rmdir` command.

To delete a directory with the rmdir command, you must have write and execute permission on the parent directory. For example, to issue the following command, you would need to have write and execute permission in the /etc directory: rmdir /etc/test

Chapter 5: File Manipulation

This chapter will cover the following exam objectives:

103.3: Perform basic file management

Weight: 4

Description: Candidates should be able to use the basic Linux commands to manage files and directories.

Key Knowledge Areas:

- Copy, move and remove files and directories individually
  Section 5.1 | Section 5.5 | Section 5.6 | Section 5.7 | Section 5.8 | Section 5.9
- Copy multiple files and directories recursively
  Section 5.5
- Remove files and directories recursively
  Section 5.7 | Section 5.9
- Use simple and advanced wildcard specifications in commands
  Section 5.5 | Section 5.7

Chapter 5: File Manipulation

cp

Command used to copy files and directories. cp will copy a SOURCE to a DEST, or multiple SOURCES to a DIRECTORY.
Section 5.5

file

Command used to determine the type of file. file tests each argument in an attempt to classify it. There are three sets of tests, preformed in this order: filesystem test, magic tests, and language tests.
Section 5.3

ls

Command that will list information about files. The current directory is listed by default.
| Section 5.2

mkdir

Command used to create directories, if they do not already exist.
Section 5.8

mv

Command that can move files from one location to another, as well as renaming files.
Section 5.6

rm

Command used to remove files or directories. By default the rm command will not remove directories.
Section 5.7

rmdir

Command that is used to remove empty directories in the filesystem.
Section 5.9

touch

Command used to change the file timestamps. touch will allow a user to update the access and modification times of each FILE to the current time.
Section 5.4