System Level Programing

Assignment 01

W.B.P.L.Wickramasinghe
CST/13/0046

chmod command

The command name chmod stands for **change mode**, and it is used to define the way a file can be accessed or to change the permissions of files or directories.

chmod command take the form;

chmod options permissions filename

If no options are specified, chmod modifies the permissions of the file specified by filename to the permissions specified by permissions.

Permissions defines the permissions for the owner of the file, members of the group who owns the file, and anyone else. There are two ways to represent these permissions: with symbols or with octal numbers.

chmod syntax:

- chmod [option] mode file
- chmod [option] octal-mode file
- chmod [option] –reference= refile file

Options:

-c, --changes	Like **--verbose**, but gives verbose output only when a change is actually made.
-f, --silent, --quiet	Quiet mode; suppress most error messages.
-v, --verbose	Verbose mode; output a diagnostic message for every file processed.
--no-preserve-root	Do not treat '**/**' (the root directory) in any special way. This is the default.
--preserve-root	Do not operate recursively on '**/**'.
--reference=refile	Set permissions to match those of file **refile**, ignoring any specified **mode**.
-R, --recursive	Change files and directories recursively.
--help	Display a help message and exit.
--version	Output version information and exit.

chmod changes the file mode of each specified **FILE** according to **MODE**, which can be either a symbolic representation of changes to make, or an octal number representing the bit pattern for the new mode bits.

chown command

The **chown** command changes the owner and owning group of files.
chown changes the user and/or group ownership of each given file. If only an owner is given, that user is made the owner of each given file, and the files' group is not changed. If the owner is followed by a colon and a group name, with no spaces between them, the group ownership of the files is changed as well. If a colon but no group name follows the user name, that user is made the owner of the files and the group of the files is changed to that user's login group. If the colon and group are given, but the owner is omitted, only the group of the files is changed; in this case, **chown** performs the same function as **chgrp**. If only a colon is given, or if the entire operand is empty, neither the owner nor the group is changed.

chown sysntax:

- chown [option] [owner] [:[group]] file
- chown [option] –reference= refile file

Options

Change the owner and/or group of each **file** to **owner** and/or **group**. With **--reference**, change the owner and group of each **file** to those of **refile**.

-c, --changes	Like **verbose** but report only when a change is made.
-f, --silent, --quiet	Suppress most error messages.
-v, --verbose	Output a diagnostic for every file processed.
--dereference	Affect the referenced file of each symbolic link rather than Symbolic link itself. This is the default.
-h, --no-dereference	Affect symbolic links instead of any referenced file. This is useful only on systems that can be changed the ownership of symlinks.

--no-preserve-root	Do not treat '**/**' (the root directory) in any special way. This is the default.
--preserve-root	Do not operate recursively on '**/**'.
--reference=refile	use RFILE's owner and group rather than specifying **owner: group** values
-R, --recursive	Operate on files and directories recursively.

Owner is unchanged if unspecified. Group is unchanged if unspecified, or changed to the login group if implied by a '**:**' following a symbolic **owner**. **owner** and **group** may be numeric as well as symbolic.

unlink() command

unlink command call the unlink function to remove the specified file.

unlink() syntax:

- unlink file
- unlink option

Options

--help	Display a help message and exit.
--version	Output version information and exit.

link() command

Calls the link function to create a link to a file.
The **link** command creates a hard link named **file2** which shares the same index node as the existing file **file1**. Since **file1** and **file2** share the same index node, they will point to the same data on the disk, and modifying one will be functionally the same as modifying the other.
This is distinct from creating a "soft" symbolic link to a file, which creates its own index node and therefore does not directly point to the same data.

For example, a user cannot create a hard link which links to a directory; but this can be accomplished using a symbolic link.

link syntax:

- Link file1 file2
- Link option

Options

--help	Display help and exit.
--version	Display version information, and exit.

mkdir command

Short for "make directory", **mkdir** is used to create directories on a file system.
If the specified **directory** does not already exist, **mkdir** creates it. More than one **directory** may be specified when calling **mkdir**.

mkdir syntax:

- mkdir [option] [directory]

Options

-m, --mode=*MODE*	Set file mode (as with the chmod command).
-p, --parents	Create parent directories as necessary. When this option is used, no error specified *DIRECTORY* already exists.
-v, --verbose	Verbose output; print a message for each created directory.

-Z, --context=*CTX*	Set the SELinux security context of each created directory to the context *CTX*.
--help	Display a help message, and exit.
--version	Display version information, and exit.

rmdir command

rmdir is removes a directory. The **rmdir** utility removes the directory entry specified by each directory argument, provided the directory is empty. Arguments are processed in the order given. To remove both a parent directory and a sub directory of that parent, the sub directory must be specified first so the parent directory is empty when **rmdir** tries to remove it.

rmdir syntax:

- rmdir [-p] directory

Options

--ignore-fail-on-non-empty	Ignore any failure which occurs solely because a directory is non-empty.
-p	Each directory argument is treated as a pathname of which all components will be remove starting with the last most component.
-v, --verbose	Display verbose information for every directory processed.
--help	Display a help message, and exit.
--version	Output version information, and exit.

chdir command

chdir is the system function for changing the current working directory. On some systems, **chdir** may also be an alias for the shell command **cd**. **chdir** changes the current working directory of the calling process to the directory specified in **path**.

chdir syntax:

- int chdir(const char * path);

If success zero (**0**) is returned. On an error, **-1** is returned, and **errno** is set appropriately.

getcwd command

getcwd is get current working directory. The **getcwd**() function copies an absolute pathname of the current working directory to the array pointed to by *buf*, which is of length *size*.

If the current absolute path name would require a buffer longer than *size* elements, **-1** is returned, and *errno* is set to **ERANGE**; an application should check for this error, and allocate a larger buffer if necessary.

If *buf* is NULL, the behaviour of **getcwd**() is undefined.

getcwd syntax:

- long getcwd(char **buf*, unsigned long *size*);

Return value

-1 on failure, with *errno* set accordingly, and the number of characters stored in *buf* on success. The contents of the array pointed to by*buf* is undefined on error.

This return value differs from the **getcwd**() library function, which returns **NULL** on failure and the address of *buf* on success.

opendir command

opendir is open a directory._The opendir() function opens a directory stream corresponding to the directory name, and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory.

The opendir() function return a pointer to the directory stream. On error, NULL is returned, and *errno* is set appropriately.

opendir syntax:

- DIR *opendir(const char *name);

The underlying file descriptor of the directory stream can be obtained using **dirfd**().

The opendir() function sets the close-on-exec flag for the file descriptor underlying the DIR *.

readdir command

readdir is read directory entry. readdir() reads one **dirent** structure from the directory pointed at by **fd** into the memory area pointed to by **dirp**. The parameter **count** is ignored; at most one dirent structure is read.

On success, 1 is returned. On end of directory, 0 is returned. On error, -1 is returned, and ***errno*is** set appropriately.

Readdir syntax:

- int readdir(unsigned int *fd*, struct dirent **dirp*, unsigned int *count*);

telldir command

telldir is return current location in directory stream. The telldir() function returns the current location associated with the directory stream **dirp**.

On success, the telldir() function returns the current location in the directory stream. On error, -1 is returned, and **errno** is set appropriately.

telldir syntax:

- long telldir(DIR*dirp);

seekdir command

seekdir is set position of directory stream. The seekdir() function sets the position of the next readdir() operation on the directory stream specified by **dirp** to the position specified by **loc**. The value of **loc** should have been returned from an earlier call to **telldir().** The new position reverts to the one associated with the directory stream when **telldir()** was performed.

If the value of **loc** was not obtained from an earlier call to **telldir()** or if a call to **rewinddir()** occurred between the call to **telldir()** and the call to **seekdir(),** the results of subsequent calls **toreaddir()** are unspecified.

The **seekdir()** function returns no value.

seekdir syntax:

void seekdir(DIR *dirp*, long int *loc*);

closedir command

closedir is close a directory._The closedir() function closes the directory stream associated with **dirp.** A successful call to closedir() also closes the underlying file descriptor associated with **dirp**. The directory stream descriptor **dirp** is not available after this call.

The closedir() function returns 0 on success. On error, -1 is returned, and **errno** is set appropriately.

closedir syntax:

- int closedir(DIR *dirp);

References

http://www.computerhope.com/unix/

https://linux.die.net/man/1/

http://www.tutorialspoint.com/unix_commands/