FILE HANDLING UTILITIES:

cat Command:

cat linux command concatenates files and print it on the standard output.

SYNTAX:

The Syntax is cat [OPTIONS] [FILE]...

OPTIONS:

- -A Show all.
- -b Omits line numbers for blank space in the output.
- -e A $ character will be printed at the end of each line prior to a new line.
- -E Displays a $ (dollar sign) at the end of each line.
- -n Line numbers for all the output lines.
- -s If the output has multiple empty lines it replaces it with one empty line.
- -T Displays the tab characters in the output.
- -v Non-printing characters (with the exception of tabs, new-lines and formfeeds) are printed visibly.

EXAMPLE:

1. To Create a new file: cat > file1.txt This command creates a new file file1.txt. After typing into the file press control+d (^d) simultaneously to end the file.

2. To Append data into the file: cat >> file1.txt To append data into the same file use append operator >> to write into the file, else the file will be overwritten (i.e., all of its contents will be erased).
3. To display a file: cat file1.txt This command displays the data in the file.
4. To concatenate several files and display: cat file1.txt file2.txt
   The above cat command will concatenate the two files (file1.txt and file2.txt) and it will display the output in the screen. Some times the output may not fit the monitor screen. In such situation you can print those files in a new file or display the file using less command. cat file1.txt file2.txt | less
5. To concatenate several files and to transfer the output to another file. cat file1.txt file2.txt > file3.txt .In the above example the output is redirected to new file file3.txt. The cat command will create new file file3.txt and store the concatenated output into file3.txt.

rm Command:

rm linux command is used to remove/delete the file from the directory.

SYNTAX:

The Syntax is s rm [options..] [file | directory]

OPTIONS:

- ❖ -f Remove all files in a directory without prompting the user.
- ❖ -i Interactive. With this option, rm prompts for confirmation before removing any files.
- ❖ -r (or) -R Recursively remove directories and subdirectories in the argument list. The directory will be emptied of files and removed. The user is normally prompted for removal of any write-protected files which the directory contains.

EXAMPLE:

1. To Remove / Delete a file: rm file1.txt Here rm command will remove/delete the file file1.txt.
2. To delete a directory tree: rm -ir tmp This rm command recursively removes the contents of all subdirectories of the tmp directory, prompting you regarding the removal of each file, and then removes the tmp directory itself.
3. To remove more files at once rm file1.txt file2.txt rm command removes file1.txt and file2.txt files at the same time.

cd Command:

cd command is used to change the directory.

SYNTAX:

The Syntax is cd [directory | ~ | ./ | ../ | -] OPTIONS:

- ❖ -L Use the physical directory structure.
- ❖ -P Forces symbolic links.

EXAMPLE:

1. cd linux-command This command will take you to the sub-directory(linux-command) from its parent directory.
2. cd .. This will change to the parent-directory from the current working directory/subdirectory.
3. cd ~ This command will move to the user's home directory which is "/home/username".

cp Command:

cp command copy files from one location to another. If the destination is an existing file, then the file is overwritten; if the destination is an existing directory, the file is copied into the directory (the directory is not overwritten).

SYNTAX: The Syntax is cp [OPTIONS]... SOURCE DEST
cp [OPTIONS]... SOURCE... DIRECTORY

cp[OPTIONS]...--target-directory=DIRECTORY SOURCE...

OPTIONS:

- -a same as
- -dpR. --backup[=CONTROL] make a backup of each existing destination file
- -b like --backup but does not accept an argument.
- -f if an existing destination file cannot be opened, remove it and try again.
- -p same as --preserve=mode,ownership,timestamps. -- preserve[=ATTR_LIST] preserve the specified attributes (default: mode,ownership,timestamps) and security contexts, if possible additional attributes: links, all. --nopreserve=ATTR_LIST don't preserve the specified attribute. --parents append source path to DIRECTORY.

EXAMPLE:

1. Copy two files: cp file1 file2 The above cp command copies the content of file1.php to file2.php.
2. To backup the copied file: cp -b file1.php file2.php Backup of file1.php will be created with '~' symbol as file2.php~.
3. Copy folder and subfolders: cp -R scripts scripts1 The above cp command copy the folder and subfolders from scripts to scripts1.

ls Command:

ls command lists the files and directories under current working directory.

SYNTAX:

The Syntax is ls [OPTIONS]... [FILE] OPTIONS:

- -l Lists all the files, directories and their mode, Number of links, owner of the file, file size, Modified date and time and filename.
- -t Lists in order of last modification time.
- -a Lists all entries including hidden files.
- -d Lists directory files instead of contents.
- -p Puts slash at the end of each directories.
- -u List in order of last access time.
- -i Display inode information.
- -ltr List files order by date.
- -lSr List files order by file size.

EXAMPLE:

1. Display root directory contents: ls / lists the contents of root directory.
2. Display hidden files and directories: ls -a lists all entries including hidden files and directories.
3. Display inode information: ls -i
   - 7373073 book.gif
   - 7373074 clock.gif
   - 7373082 globe.gif
   - 7373078 pencil.gif
   - 7373080 child.gif
   - 7373081 email.gif
   - 7373076 indigo.gif

The above command displays filename with inode value.

ln Command:

ln command is used to create link to a file (or) directory. It helps to provide soft link for desired files. Inode will be different for source and destination.

SYNTAX:

The Syntax is ln [options] existingfile(or directory)name newfile(or directory)name OPTIONS:

- -f Link files without questioning the user, even if the mode of target forbids writing. This is the default if the standard input is not a terminal.
- -n Does not overwrite existing files.

- -s Used to create soft links.

EXAMPLE:

1. ln -s file1.txt file2.txt Creates a symbolic link to 'file1.txt' with the name of 'file2.txt'. Here inode for 'file1.txt' and 'file2.txt' will be different.
2. ln -s nimi nimi1 Creates a symbolic link to 'nimi' with the name of 'nimi1'.

mkdir COMMAND:

mkdir command is used to create one or more directories.

SYNTAX: The Syntax is mkdir [options] directories OPTIONS:

- -m Set the access mode for the new directories.
- -p Create intervening parent directories if they don't exist.
- -v Print help message for each directory created.

EXAMPLE:

1. Create directory: mkdir test The above command is used to create the directory 'test'.
2. Create directory and set permissions:
   mkdir -m 666 test The above command is used to create the directory 'test' and set the read and write permission.

rmdir Command:

rmdir command is used to delete/remove a directory and its subdirectories.

SYNTAX:

The Syntax is rmdir [options..] Directory

OPTIONS:

- -p Allow users to remove the directory dirname and its parent directories which become empty.

EXAMPLE:

1. To delete/remove a directory rmdir tmp rmdir command will remove/delete the directory tmp if the directory is empty.
2. To delete a directory tree: rm -ir tmp This command recursively removes the contents of all subdirectories of the tmp directory, prompting you regarding the removal of each file, and then removes the tmp directory itself.

mv Command:

mv command which is short for move. It is used to move/rename file from one directory to another. mv command is different from cp command as it completely removes the file from the source and moves to the directory specified, where cp command just copies the content from one file to another.

SYNTAX:

The Syntax is mv [-f] [-i] oldname newname

OPTIONS:

- -f This will not prompt before overwriting (equivalent to --reply=yes). mv -f will move the file(s) without prompting even if it is writing over an existing target. -i Prompts before overwriting another file.

EXAMPLE:

1. To Rename / Move a file: mv file1.txt file2.txt This command renames file1.txt as file2.txt
2. To move a directory mv hscripts tmp In the above line mv command moves all the files, directories and sub-directories from hscripts folder/directory to tmp directory if the tmp directory already exists. If there is no tmp directory it rename's the hscripts directory as tmp directory.
3. To Move multiple files/More files into another directory mv file1.txt tmp/file2.txt newdir This command moves the files file1.txt from the current directory and file2.txt from the tmp folder/directory to newdir.

diff **Command:**

diff command is used to find differences between two files.

SYNTAX: The Syntax is diff [options..] from-file to-file

OPTIONS:

- -a Treat all files as text and compare them line-by-line.
- -b Ignore changes in amount of white space.
- -c Use the context output format.
- -e Make output that is a valid ed script.
- -H Use heuristics to speed handling of large files that have numerous scattered small changes.
- -i Ignore changes in case; consider upper- and lower-case letters equivalent.
- -n Prints in RCS-format, like -f except that each command specifies the number of lines affected.
- -q Output RCS-format diffs; like -f except that each command specifies the number of lines affected.
- -r When comparing directories, recursively compare any subdirectories found.
- -s Report when two files are the same.
- -w Ignore white space when comparing lines.
- -y Use the side by side output format.

EXAMPLE:

Lets create two files file1.txt and file2.txt and let it have the following data. Data in file1.txt Data in file2.txt HIOX TEST hscripts.com with friend ship HIOX TEST HSCRIPTS.com with friend ship hiox india

1. Compare files ignoring white space: diff -w file1.txt file2.txt This command will compare the file file1.txt with file2.txt ignoring white/blank space and it will produce the following output. 2c2 < hscripts.com --- > HSCRIPTS.com 4d3 < Hioxindia.com
2. Compare the files side by side, ignoring white space: diff -by file1.txt file2.txt This command will compare the files ignoring white/blank space, It is easier to differentiate the files. HIOX TEST HIOX TEST hscripts.com | HSCRIPTS.com with friend ship with friend ship Hioxindia.com < The third line(with friend ship) in file2.txt has more blank spaces, but still the -b ignores the blank space and does not show changes in the particular line, -y printout the result side by side.
3. Compare the files ignoring case. diff -iy file1.txt file2.txt This command will compare the files ignoring case(upper-case and lower-case) and displays the following output. HIOX TEST HIOX TEST hscripts.com HSCRIPTS.com
    - About wc Short for word count.wc displays a count of lines, words, and characters in a file. Syntax wc [-c | -m | -C] [-l] [-w] [file ...] -c Count bytes. -m Count characters. -C Same as -m. -l Count lines.
    - -w Count words delimited by white space characters or new line characters. Delimiting characters are Extended Unix Code (EUC) characters from any code set defined by iswspace() File Name of file to word count.
      Examples wc myfile.txt - Displays information about the file myfile.txt. Below is an example of the output.
      5 13 57 myfile.txt 5 = Lines 13 = Words 57 = Characters About split Split a file into pieces. Syntax split [-linecount | -l linecount] [-a suffixlength] [file [name]] split -b n [k | m] [-a suffixlength] [file [name]] -linecount | -l linecount Number of lines in each piece. Defaults to 1000 lines. -a suffixlength Use suffixlength letters to form the suffix portion of the filenames of the split file. If -a is not specified, the default suffix length is 2. If the sum of the name operand and the suffixlength option-argument would create a filename exceeding NAME_MAX bytes, an error will result; split will exit with a diagnostic message and no files will be created. -b n Split a file into pieces n bytes in size. -b n k Split a file into pieces n*1024 bytes in size. -b n m Split a file into pieces n*1048576 bytes in size. File The path name of the ordinary file to be split. If no input file is given or file is -, the standard input will be used. name The prefix to be used for each of the files resulting from the split operation. If no name argument is given, x will be used as the prefix of the output files. The combined length of the basename of prefix and suffixlength cannot exceed NAME_MAX bytes; see OPTIONS.

Examples split -b 22 newfile.txt new - would split the file "newfile.txt" into three separate files called newaa, newab and newac each file the size of 22. split -l 300 file.txt new - would split the file "newfile.txt" into files beginning with the name "new" each containing 300 lines of text each About settime and touch Change file access and modification time.

Syntax touch [-a] [-c] [-m] [-r ref_file | -t time] file settime [-f ref_file] file -a Change the access time of file. Do not change the modification time unless - m is also specified. -c Do not create a specified file if it does not exist. Do not write any diagnostic messages concerning this condition. -m Change the modification time of file. Do not change the access time unless -a is also specified. -r ref_file Use the corresponding times of the file named by ref_file instead of the current time. -t time Use the specified time instead of the current time. time will be a decimal number of the form: [[CC]YY]MMDDhhmm [.SS] MM - The month of the year [01-12]. DD - The day of the month [01-31]. hh - The hour of the day [00-23]. mm - The minute of the hour [00-59]. CC - The first two digits of the year. YY - The second two digits of the year. SS - The second of the minute [00-61]. -f ref_file Use the corresponding times of the file named by ref_file instead of the current time. File A path name of a file whose times are to be modified. Examples settime myfile.txt Sets the file myfile.txt as the current time / date. touch newfile.txt Creates a file known as "newfile.txt", if the file does not already exist. If the file already exists the accessed / modification time is updated for the file newfile.txt About comm Select or reject lines common to two files. Syntax comm [-1] [-2] [-3] file1 file2 -1 Suppress the output column of lines unique to file1. -2 Suppress the output column of lines unique to file2. -3 Suppress the output column of lines duplicated in file1 and file2. file1 Name of the first file to compare.

file2 Name of the second file to compare. Examples comm myfile1.txt myfile2.txt The above example would compare the two files myfile1.txt and myfile2.txt.

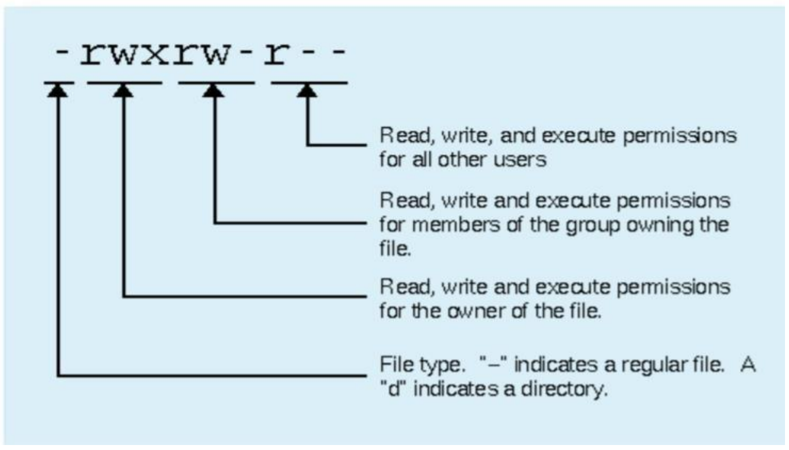
Security By File Permissions

This lesson will cover the following commands:

- chmod - modify file access rights
- su - temporarily become the superuser •
- chown - change file ownership
- chgrp - change a file's group ownership File permissions Linux uses the same permissions scheme as Linux. Each file and directory on your system is assigned access rights for the owner of the file, the members of a group of related users, and everybody else. Rights can be assigned to read a file, to write a file, and to execute a file (i.e., run the file as a program).

To see the permission settings for a file, we can use the ls command as follows: [me@linuxbox me]$ ls -l some_file -rw-rw-r-- 1 me me 1097374 Sep 26 18:48 some_file We can determine a lot from examining the results of this command:

- The file "some_file" is owned by user "me"
- User "me" has the right to read and write this file
- The file is owned by the group "me"
- Members of the group "me" can also read and write this file
- Everybody else can read this file Let's try another example. We will look at thebash program which is located in the /bin directory: [me@linuxbox me]$ ls -l /bin/bash -rwxr-xr-x 1 root root 316848 Feb 27 2000 /bin/bash Here we can see:
- The file "/bin/bash" is owned by user "root"
- The superuser has the right to read, write, and execute this file
- The file is owned by the group "root"
- Members of the group "root" can also read and execute this file
- Everybody else can read and execute this file In the diagram below,


-rwxrw-r--
Read, write, and execute permissions for all other users
Read, write and execute permissions for members of the group owning the file.
Read, write and execute permissions for the owner of the file.
File type. "–" indicates a regular file. A "d" indicates a directory.

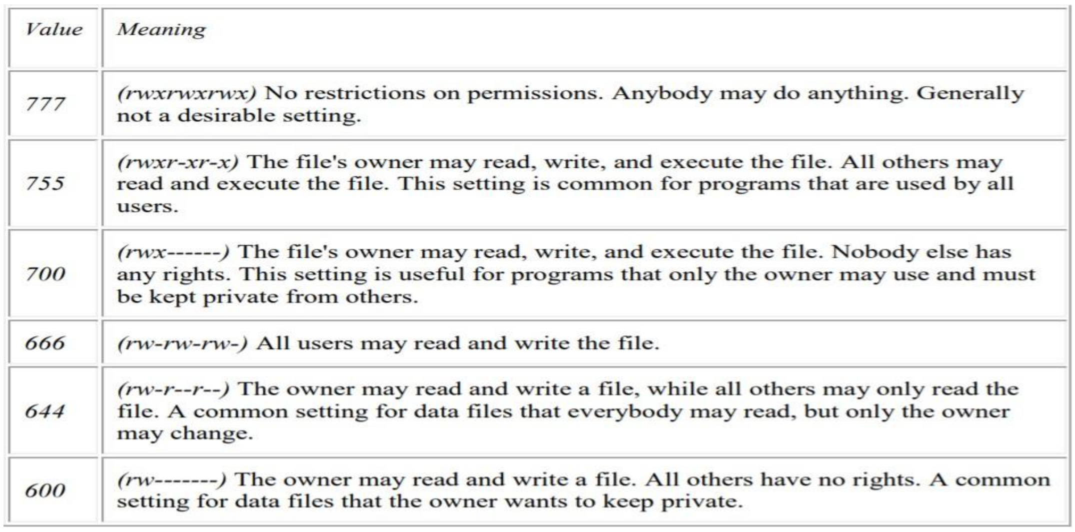
we see how the first portion of the listing is interpreted. It consists of a character indicating the file type, followed by three sets of three characters that convey the reading, writing and execution permission for the owner, group, and everybody else. chmod The chmod command is used to change the permissions of a file or directory. To use it, you specify the desired permission settings and the file or files that you wish to modify. There are two ways to specify the permissions, but I am only going to teach one way. It is easy to think of the permission settings as a series of bits (which is how the computer thinks about them).

Here's how it works:

rwx rwx rwx = 111 111 111
rw- rw- rw- = 110 110 110
rwx------- = 111 000 000
and so on...
rwx = 111 in binary = 7
rw- = 110 in binary = 6
r-x = 101 in binary = 5
r-- = 100 in binary = 4

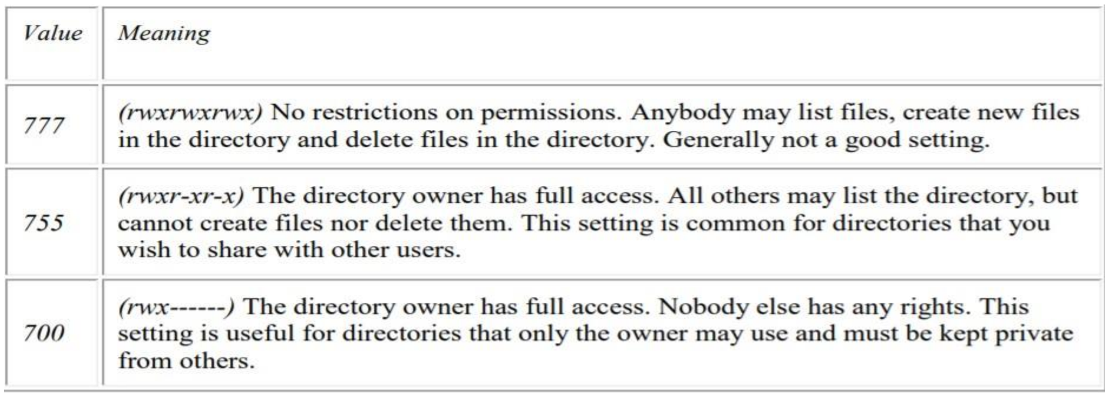
Now, if you represent each of the three sets of permissions (owner, group, and other) as a single digit, you have a pretty convenient way of expressing the possible permissions settings. For example, if we wanted to set some_file to have read and write permission for the owner, but wanted to keep the file private from others, we would:

[me@linuxbox me]$ chmod 600 some_file Here is a table of numbers that covers all the common settings. The ones beginning with "7" are used with programs (since they enable execution) and the rest are for other kinds of files.



Value	*Meaning*
777	*(rwxrwxrwx)* No restrictions on permissions. Anybody may do anything. Generally not a desirable setting.
755	*(rwxr-xr-x)* The file's owner may read, write, and execute the file. All others may read and execute the file. This setting is common for programs that are used by all users.
700	*(rwx------)* The file's owner may read, write, and execute the file. Nobody else has any rights. This setting is useful for programs that only the owner may use and must be kept private from others.
666	*(rw-rw-rw-)* All users may read and write the file.
644	*(rw-r--r--)* The owner may read and write a file, while all others may only read the file. A common setting for data files that everybody may read, but only the owner may change.
600	*(rw-------)* The owner may read and write a file. All others have no rights. A common setting for data files that the owner wants to keep private.

Directory permissions The chmod command can also be used to control the access permissions for directories. In most ways, the permissions scheme for directories works the same way as they do with files. However, the execution permission is used in a different way. It provides control for access to file listing and other things. Here are some useful settings for directories:



Value	*Meaning*
777	*(rwxrwxrwx)* No restrictions on permissions. Anybody may list files, create new files in the directory and delete files in the directory. Generally not a good setting.
755	*(rwxr-xr-x)* The directory owner has full access. All others may list the directory, but cannot create files nor delete them. This setting is common for directories that you wish to share with other users.
700	*(rwx------)* The directory owner has full access. Nobody else has any rights. This setting is useful for directories that only the owner may use and must be kept private from others.