**Files**

Table of Contents

[Listing Files 2](#_Toc83133450)

[Metacharacters 3](#_Toc83133451)

[Dots 3](#_Toc83133452)

[File Management 4](#_Toc83133453)

[File Permissions 5](#_Toc83133454)

[Users and Permissions 5](#_Toc83133455)

[Changing Permissions 6](#_Toc83133456)

[Simplifying Permission Changes 7](#_Toc83133457)

[Changing Owners and Groups 7](#_Toc83133458)

## Listing Files

*$* ls # Lists the contents of the current directory  
*$* ls -l # Same as previous; shows more information  
*$* ls -a # Same as ls; lists hidden items as well  
*$* ls -la # Same as previous; shows more information

SHELL

For the **ls -l** and **ls -la** commands, the information displayed will come in the form of **seven columns**.

1. File type and permissions
2. Number of memory blocks occupied
3. Owner
4. Owner group
5. File size (in bytes)
6. Date and time of last modification
7. File name

For every file, the filetype begins with one of **seven characters**.

|  |  |
| --- | --- |
| **-** | Regular file, such as an ASCII text file, binary executable, or hard link. |
| **b** | Block special file. Block input/output device file such as a physical HDD. |
| **c** | Character special file. Raw input/output device file such as a physical HDD. |
| **d** | Directory file that contains a listing of other files and directories. |
| **l** | Symbolic link file. Links on any regular file. |
| **p** | Named pipe. A mechanism for inter-process communications. |
| **s** | Socket used for inter-process communication. |

### Metacharacters

We can also use the **ls** command to search for **specific files** or combine the search with **metacharacters** to make powerful searches.

The **\*** metacharacter matches with or more characters.

The **?** metacharacter matches with a single character.

*$* ls ch\*.docx # Searches for any files in the current directory  
 # that starts with ch and ends with .docx  
 # with 0 or more characters in between

*$* ls ch?.docx # Same as above; there can only be one character in between

SHELL

### Dots

**Dots** can frequently be found when listing files and directories.

A single dot, **.**, refers to the **current directory**, while two dots, **..**, indicate the **parent directory**. We can use these with the **ls** command, or with any other compatible command.

*$* ls . # Lists the files in the current directory  
*$* ls .. # Lists the files in the parent directory

SHELL

## File Management

*$* touch newFile.txt # Creates a new file in the current directory  
*$* vi newFile.txt # Same as above  
 # also opens the file in the built-in vi editor  
 # opens the file if it already exists instead of  
 # creating a new one  
  
*$* cat existingFile.txt # Displays the content of the file in the terminal  
*$* cat -b existingFile.txt # Same as above; also shows line numbers  
  
*$* wc existingFile.txt # Displays the number of lines,  
 # words and characters in the file  
*$* wc file1.txt file2.txt file3.txt # Same as above; works with multiple files  
  
*$* cp existingFile.txt newFile.txt # Creates a copy of an existing file  
 # replaces file if it exists  
*$* mv existingFile.txt newFile.txt # Renames an existing file  
 # technically, moves it  
 # replaces file it is exists  
  
*$* rm existingFile.txt # Removes an existing file  
*$* rm -i existingFile.txt # Same as above; shows warning  
*$* rm file1.txt file2.txt file3.txt # Removes multiple files

SHELL

## File Permissions

When we run the command **ls -l** on the Linux shell, we get a list of the files in the current directory, along with some extra information about the files. For example, one line of the results could look like this:

-rwxr--r-- 1 user user 0 Jun 2 09:26 test.txt

SHELL

We will be concentrating on the first part of these results for now, the part that says -rwxr--r--.

As discussed before, the first character, -, tells us that this is a regular file. The other nine character are what we are interested right now. These nine characters are divided into **three parts** of **three characters** each, rwx, r--and r--.

### Users and Permissions

Each part refers to **file permissions** for a category of users. The first part is for the **owner** of the file, the second part is for the users in the same **group** that the file belongs to and the third part is for **other** users.

Permissions are indicated by one of three characters. **r** refers to **read**, **w** refers to **write** and **x** refers to **execute**. If a particular permission is not given for a user, the character is replaced with a **-**.

Thus, in the example we saw above, the owner permission are read, write and execute and the group permissions and other permissions are both read only, not write or execute.

### Changing Permissions

We can change permissions for a file using one of three operators. The **+** operator is used to add a permission, the **–** operator is used to remove a permission and the **=** operator is used to assign a permission.

For example, say we want to:

1. Remove the write and execute permissions for the owner.
2. Add the execute permission for the group.
3. Remove the read permission for the group.
4. Assign the write permission to the owner.

The operations for each of these would be u-wx, g+x, g-r and o=w respectively. The order is irrelevant. The command used to perform the operations is:

*$* chmod u-wx,g+x,g-r,o=w test.txt

SHELL

It may be confusing what the difference between the **+** and the **=** operators is. The **+** operator **adds** permissions on top of existing ones. When we write g+x, we are adding the execute permissions without removing the existing read permissions. The **=** operator **replaces** permissions. Whatever permissions we assign is what the permissions for that user type will be. Thus, o=w means other permissions are set to write-only now. The read permissions they used to have are gone. The above information also means we can **combine** the two operations, g+x and g-r into just g=x. Thus, the command becomes:

*$* chmod u-wx,g=x,o=w test.txt

SHELL

### Simplifying Permission Changes

Changing permissions in the way we just saw is a bit cumbersome. We could simplify the process by using **binary values**.

Suppose we use **3 bits** to represent the read, write and execute permissions respectively. To allow a certain permission, we turn the bit on and to not allow it, we turn the bit off. For example, the permission r-x would be 101.

We can use the decimal equivalent for the binary number we create to set permissions for each user type. In the example above, we saw that the owner permissions were set to just read, which is 100, or 4 in decimal, the group permissions were set to execute, which is 001, or 1 in decimal and the other permissions were set to write, which is 010 or 2 in decimal. Thus, the command to set these permissions would be:

*$* chmod 412 test.txt

SHELL

### Changing Owners and Groups

We can change the **owner** of a file using the **chown** command.

*$* chown someUser test.txt

SHELL

We can change the **group** of a file using the **chgrp** command.

*$* chgrp someGroup test.txt

SHELL