10 example of chmod command in UNIX Linux

javarevisited.blogspot.in/2012/03/10-example-of-chmod-command-in-unix.html

chmod command in UNIX or Linux is used to change file or directory permissions. This is one of many UNIX basic commands which a UNIX or Linux user must be familiar with. In this UNIX command tutorial we will see **how to change file permissions using chmod command**, what are file permissions in UNIX, how to change permissions of directory and sub-directory using UNIX chmod command and finally how to create executable files in UNIX using chmod command. Before going directly into examples of *chmod command* let's spend few minutes on understanding file permissions in UNIX and why do we need to change file permissions etc.

In UNIX each file has three permission read, write and execute and three classes user(owner), group and others. So each file is provided permissions in combination of class and permissions. You can see the permission of individual file or directory by using ls command. For example in below file

```
example@localhost~/test ls -lrt stock_trading_systems
-rwxrwxrwx 1 example Domain Users 0 Jul 15 11:42 stock_trading_systems*
```

It has read, write and execute permission for all users, group and others. You can read more about file permissions in UNIX in my post beginner's guide to UNIX file permissions. To read more about UNIX and Linux file permissions see this UNIX tutorial on file and directory permissions.

Chmod command Examples in UNIX and Linux

Now let's see some practical and frequently used **example of chmod command in UNIX**

chmod command Example 1: making read only file in Unix

In this example of chmod command in UNIX we will see how to make a file read only by only providing read access to owner. You can also give read access to group and others and keep write access for owner which we will see in subsequent examples.

```
example@localhost~/test ls -lrt stock_trading_systems
-rwxrwxrwx 1 example Domain Users 0 Jul 15 11:42 stock_trading_systems*
```

Here file stock_trading_systems has read, write and execute permission "-rwxrwxrwx" for all

```
example@localhost~/test chmod 400 stock_trading_systems
```

400 means 100 000 000 means r-- --- --- i.e. read only for owner

```
example@localhost~/test ls -lrt stock_trading_systems
-r-------- 1 example Domain Users 0 Jul 15 11:42 stock_trading_systems
```

Now file is read only and only owner can read it " -r--------"

chmod command Example 2: change permissions only for user, group or others.

In this example of chmod command we will see how to change file permissions on user, group and others level. You can easily modify file permission for any of these classes. If you are using text format than "u" is user , "o" is other and "g" is group also "r" is read , "w" is write and "x" is execute. + means adding permission and "-"is removing permission.

```
example@localhost~/test ls -lrt chmod_examples
-r-------- 1 example Domain Users 0 Jul 15 11:42 chmod_examples

example@localhost~/test chmod u+w chmod_examples

example@localhost~/test ls -lrt chmod_examples
-rw------- 1 example Domain Users 0 Jul 15 11:42 chmod_examples
```

Now let's change file permissions only for group by using chmod command

```
example@localhost~/test ls -lrt chmod_examples
-rw------- 1 example Domain Users 0 Jul 15 11:42 chmod_examples

example@localhost~/test chmod g+w chmod_examples

example@localhost~/test ls -lrt chmod_examples
-rw--w---- 1 example Domain Users 0 Jul 15 11:42 chmod_examples
```

In this chmod command example we will change permission only for others class without affecting user and group class.

```
example@localhost~/test ls -lrt chmod_examples
-rw--w---- 1 example Domain Users 0 Jul 15 11:42 chmod_examples

example@localhost~/test chmod o+w chmod_examples

example@localhost~/test ls -lrt chmod_examples
-rw--w--w- 1 example Domain Users 0 Jul 15 11:42 chmod_examples
```

Chmod command Example 3: change file permissions for all (user + group + others)

In last unix chmod example we learn how to change permission for user, group and others individually but some time its convenient to change permissions for all instead of modifying individual permission for user, group and other. If you are providing permission in text format than "a" is used for "all" while "u" is used for user.

```
example@localhost~/test ls -lrt linux_command.txt
-rw--w--w- 1 example Domain Users 0 Jul 15 11:42 linux_command.txt

example@localhost~/test chmod a+x linux_command.txt

example@localhost~/test ls -lrt linux_command.txt
-rwx-wx-wx 1 example Domain Users 0 Jul 15 11:42 linux_command.txt*
```

Chmod command Example 4: Changing permissions in numeric format of chmod command

Chmod command in UNIX and Linux allows modifying permissions not just on text format which is more readable but also on numeric format where combination of permissions are represented in octal format e.g. 777 where first digit is for user, second is for group and 3rd is for others. Now if you write down 1st digit in binary format it will be written as 111 on which 1st digit is for read permission, 2nd is for write and 3rd is for execute permission.

```
example@localhost~/test ls -lrt unix_command.txt
-rw--w--w- 1 example Domain Users 0 Jul 15 11:42 unix_command.txt

example@localhost~/test chmod 777 unix_command.txt

example@localhost~/test ls -lrt unix_command.txt
-rwxrwxrwx 1 example Domain Users 0 Jul 15 11:42 unix_command.txt*
```

Chmod command Example 5: How to remove file permission using chmod command Unix

In this example of chmod command in UNIX we will see how to remove various permissions from files. You can easily remove read, write or execute permission from file using chmod command in both numeric and text format. Below examples shows removal of execute permission represented by –x in text format.

```
example@localhost~/test ls -lrt linux_command.txt
-rwx-wx-wx 1 example Domain Users 0 Jul 15 11:42 linux_command.txt*

example@localhost~/test chmod a-x linux_command.txt

example@localhost~/test ls -lrt linux_command.txt
-rw--w--w- 1 example Domain Users 0 Jul 15 11:42 linux_command.txt
```

Chmod command Example 6: changing permission for directory and subdirectory recursively in Unix

This is the most frequently used example of chmod command where we want to provide permission to any directory and all contents inside that directory including files and sub directories. By using –R command option of chmod in Unix you can provide permissions recursively to any directory as shown in below example of chmod command.

```
example@localhost~/test ls -lrt
total 8.0K
-rwxrwxrwx  1 example Domain Users    0 Jul 15 11:42 unix_command.txt*
drwxr-xr-x+ 1 example Domain Users    0 Jul 15 14:33 stocks/

example@localhost~/test chmod -R 777 stocks/

example@localhost~/test ls -lrt
total 8.0K
-rwxrwxrwx  1 example Domain Users    0 Jul 15 11:42 unix_command.txt*
drwxrwxrwx+ 1 example Domain Users    0 Jul 15 14:33 stocks/

example@localhost~/test ls -lrt stocks
total 0
-rwxrwxrwx 1 example Domain Users 0 Jul 15 14:33 online_stock_exchanges.txt*
```

Chmod command Example 7: How to remove read and write access from file for all
So far we have been seeing how to provide read, write and execute permission to file and directory in UNIX and now we will see opposite of that i.e. how to remove read, write and execute access. Its simple in text format because instead of + we are going to use -. Just like + used to add permission – will be used to remove permissions.

```
example@localhost~/test ls -lrt stock_trading_systems
-rwxrwxrwx 1 example Domain Users 0 Jul 15 11:42 stock_trading_systems*

example@localhost~/test chmod a-wx stock_trading_systems

example@localhost~/test ls -lrt stock_trading_systems
-r--r--r-- 1 example Domain Users 0 Jul 15 11:42 stock_trading_systems
```

Chmod command Example 8: setting execute permission only on directories without touching files

Many times we just want to provide directory or subdirectory execute permission without modifying permissions on file just to make those directories searchable. Until I know this command I used to do this by finding all directory and then changing there execute permission but we have a better way to do it by using chmod command in UNIX. You can use "X" (capital X) options to provide execute permission to only directories without touching files. Let’s see an example of chmod command for that:

```
example@localhost~/test ls -lrt
total 8.0K
-r--r--r--  1 example Domain Users    0 Jul 15 11:42 stock_trading_systems
drw-rw-rw-+ 1 example Domain Users    0 Jul 15 14:33 stocks/

example@localhost~/test chmod a+X *

example@localhost~/test ls -lrt
total 8.0K
-r--r--r--  1 example Domain Users    0 Jul 15 11:42 stock_trading_systems
drwxrwxrwx+ 1 example Domain Users    0 Jul 15 14:33 stocks/
```

Remember to use X (capital case) if you use x (small case) it will affect all files and directories.

Chmod command Example 9: changing mutiple permission for a file or directory in Unix or Linux
You can change combination of user + groups or groups+ other in one command to modify permissions of files and directory. Below example of chmod command just doing same it’s providing execute permission for user and read, execute permission

```
example@localhost~/test ls -lrt
total 8.0K
-r--r--r--  1 example Domain Users    0 Jul 15 11:42 stock_trading_systems
drwxrwxrwx+ 1 example Domain Users    0 Jul 15 14:33 stocks/

example@localhost~/test chmod u+x,g+x stock_trading_systems

example@localhost~/test ls -lrt stock_trading_systems
-r-xr-xr-- 1 example Domain Users 0 Jul 15 11:42 stock_trading_systems*
```

Chmod command Example 10: How to copy permission from one file to another in Unix

This is very interesting example of chmod command in UNIX which copies permission from one file to another. You can easily reference source file and copy all permissions on that file to destination file as shown in following chmod example:

```
example@localhost~/test ls -lrt future_trading
-rwxrwxrwx 1 example Domain Users 0 Jul 15 15:30 future_trading*

example@localhost~/test ls -lrt stock_trading_systems
-r--r--r-- 1 example Domain Users 0 Jul 15 11:42 stock_trading_systems

example@localhost~/test chmod --reference=stock_trading_systems future_trading

example@localhost~/test ls -lrt future_trading
-r--r--r-- 1 example Domain Users 0 Jul 15 15:30 future_trading
```

These were some of **frequently used example of chmod command in UNIX or Linux**. Chmod command is as useful as UNIX find command or grep command and knowing how to change file permissions is essential skill while working in UNIX. Please share if you have any other example of chmod command which we should be aware of.

Other **UNIX command tutorial** you may like

12 Linux Chown Command Examples to Change Owner and Group

thegeekstuff.com/2012/06/chown-examples/

The concept of owner and groups for files is fundamental to Linux. Every file is associated with an owner and a group. You can use chown and chgrp commands to change the owner or the group of a particular file or directory.

In this article, we will discuss the 'chown' command as it covers most part of the 'chgrp' command also.

Even if you already know this command, probably one of the examples mentioned below might be new to you.

1. Change the owner of a file

```
# ls -lart tmpfile
-rw-r--r-- 1 himanshu family 0 2012-05-22 20:03
tmpfile

# chown root tmpfile

# ls -l tmpfile
-rw-r--r-- 1 root family 0 2012-05-22 20:03 tmpfile
```

So we see that the owner of the file was changed from 'himanshu' to 'root'.

2. Change the group of a file

Through the chown command, the group (that a file belongs to) can also be changed.

```
# ls -l tmpfile
-rw-r--r-- 1 himanshu family 0 2012-05-22 20:03 tmpfile

# chown :friends tmpfile

# ls -l tmpfile
-rw-r--r-- 1 himanshu friends 0 2012-05-22 20:03
tmpfile
```

If you observe closely, the group of the file changed from 'family' to 'friends'. So we see that by just adding a ':' followed by the new group name, the group of the file can be changed.

3. Change both owner and the group

```
# ls -l tmpfile
-rw-r--r-- 1 root family 0 2012-05-22 20:03 tmpfile

# chown himanshu:friends tmpfile

# ls -l tmpfile
-rw-r--r-- 1 himanshu friends 0 2012-05-22 20:03
tmpfile
```

So we see that using the syntax '<newOwner>:<newGroup>', the owner as well as group can be changed in one go.

4. Using chown command on symbolic link file

Here is a symbolic link :

```
# ls -l tmpfile_symlnk
lrwxrwxrwx 1 himanshu family 7 2012-05-22 20:03 tmpfile_symlnk ->
tmpfile
```

So we see that the symbolic link 'tmpfile_symlink' links to the file 'tmpfile'.

Lets see what happens if chown command is issued on a symbolic link:

```
# chown root:friends tmpfile_symlnk

# ls -l tmpfile_symlnk
lrwxrwxrwx 1 himanshu family 7 2012-05-22 20:03 tmpfile_symlnk ->
tmpfile

# ls -l tmpfile
-rw-r--r-- 1 root friends 0 2012-05-22 20:03 tmpfile
```

When the chown command was issued on symbolic link to change the owner as well as the group then its the referent of the symbolic link ie 'tmpfile' whose owner and group got changed. This is the default behavior of the chown command. Also, there exists a flag '–dereference' for the same.

5. Using chown command to forcefully change the owner/group of symbolic file.

Using flag '-h', you can forcefully change the owner or group of a symbolic link as shown below.

```
# ls -l tmpfile_symlnk
lrwxrwxrwx 1 himanshu family 7 2012-05-22 20:03 tmpfile_symlnk ->
tmpfile

# chown -h root:friends tmpfile_symlnk

# ls -l tmpfile_symlnk
lrwxrwxrwx 1 root friends 7 2012-05-22 20:03 tmpfile_symlnk -> tmpfile
```

6. Change owner only if a file is owned by a particular user

Using chown "–from" flag, you can change the owner of a file, only if that file is already owned by a particular owner.

```
# ls -l tmpfile
-rw-r--r-- 1 root friends 0 2012-05-22 20:03 tmpfile

# chown --from=guest himanshu tmpfile

# ls -l tmpfile
-rw-r--r-- 1 root friends 0 2012-05-22 20:03 tmpfile

# chown --from=root himanshu tmpfile

# ls -l tmpfile
-rw-r--r-- 1 himanshu friends 0 2012-05-22 20:03
tmpfile
```

- In the example above, we verified that the original owner/group of the file 'tmpfile' was root/friends.
- Next we used the '–from' flag to change the owner to 'himanshu' but only if the existing owner is 'guest'.
- Now, as the existing owner was not 'guest'. So, the command failed to change the owner of the file.
- Next we tried to change the owner if the existing owner is 'root' (which was true) and this time command was successful and the owner was changed to 'himanshu'.

On a related note, if you want to change the permission of a file, you should use chmod command.

If you are a beginner, you should start by reading the basics of file permissions.

7. Change group only if a file already belongs to a certain group

Here also the flag '–from' is used but in the following way:

```
# ls -l tmpfile
-rw-r--r-- 1 himanshu friends 0 2012-05-22 20:03
tmpfile

# chown --from=:friends :family tmpfile

# ls -l tmpfile
-rw-r--r-- 1 himanshu family 0 2012-05-22 20:03 tmpfile
```

Since the file 'tmpfile' actually belonged to group 'friends' so the condition was correct and the command was successful.

So we see that by using the flag '–from=:<conditional-group-name>' we can change the group under a particular condition.

NOTE: By following the template '–from=<conditional-owner-name>:<conditional-group-name>', condition on both the owner and group can be applied.

8. Copy the owner/group settings from one file to another

This is possible by using the '–reference' flag.

```
# ls -l file
-rwxr-xr-x 1 himanshu family 8968 2012-04-09 07:10
file

# ls -l tmpfile
-rw-r--r-- 1 root friends 0 2012-05-22 20:03 tmpfile

# chown --reference=file tmpfile

# ls -l tmpfile
-rw-r--r-- 1 himanshu family 0 2012-05-22 20:03
tmpfile
```

In the above example, we first checked the owner/group of the reference-file 'file' and then checked the owner/group of the target-file 'tmpfile'. Both were different. Then we used the chown command with the '–reference' option to apply the owner/group settings from the reference file to the target file. The command was successful and the owner/group settings of 'tmpfile' were made similar to the 'file'.

9. Change the owner/group of the files by traveling the directories recursively

This is made possible by the '-R' option.

```
# ls -l linux/linuxKernel
-rw-r--r-- 1 root friends 0 2012-05-22 21:52 linux/linuxKernel

# ls -l linux/ubuntu/ub10
-rw-r--r-- 1 root friends 0 2012-05-22 21:52 linux/ubuntu/ub10

# ls -l linux/redhat/rh7
-rw-r--r-- 1 root friends 0 2012-05-22 21:52 linux/redhat/rh7

# chown -R himanshu:family linux/

# ls -l linux/redhat/rh7
-rw-r--r-- 1 himanshu family 0 2012-05-22 21:52 linux/redhat/rh7

# ls -l linux/ubuntu/ub10
-rw-r--r-- 1 himanshu family 0 2012-05-22 21:52
linux/ubuntu/ub10

# ls -l linux/linuxKernel
-rw-r--r-- 1 himanshu family 0 2012-05-22 21:52
linux/linuxKernel
```

So we see that after checking the owner/group of all the files in the directory 'linux' and its two sub-directories 'ubuntu' and 'redhat'. We issued the chown command with the '-R' option to change both the owner and group. The command was successful and owner/group of all the files was changed successfully.

10. Using chown command on a symbolic link directory

Lets see what happens if we issue the ‘chown’ command to recursively change the owner/group of files in a directory that is a symbolic link to some other directory.

Here is a symbolic link directory ‘linux_symlnk’ that links to the directory ‘linux’ (already used in example ‘9’ above) :

```
$ ls -l linux_symlnk
lrwxrwxrwx 1 himanshu family 6 2012-05-22 22:02 linux_symlnk ->
linux/
```

Now, lets change the owner (from himanshu to root) of this symbolic link directory recursively :

```
# chown -R root:friends linux_symlnk

# ls -l linux_symlnk/
-rw-r--r-- 1 himanshu friends    0 2012-05-22 21:52
linuxKernel
drwxr-xr-x 2 himanshu friends 4096 2012-05-22 21:52 redhat
drwxr-xr-x 2 himanshu friends 4096 2012-05-22 21:52 ubuntu
```

In the ouput above we see that the owner of the files and directories was not changed. This is because by default the ‘chown’ command cannot traverse a symbolic link. This is the default behavior but there is also a flag ‘-P’ for this.

11. Using chown to forcefully change the owner/group of a symbolic link directory recursively

This can be achieved by using the flag -H

```
# chown -R -H guest:family linux_symlnk

# ls -l linux_symlnk/
total 8
-rw-r--r-- 1 guest family    0 2012-05-22 21:52
linuxKernel
drwxr-xr-x 2 guest family 4096 2012-05-22 21:52 redhat
drwxr-xr-x 2 guest family 4096 2012-05-22 21:52 ubuntu
```

So we see that by using the -H flag, the owner/group of all the files/folder were changed.

12. List all the changes made by the chown command

Use the verbose option -v, which will display whether the ownership of the file was changed or retained as shown below.

```
# chown -v -R guest:friends linux
changed ownership of `linux/redhat/rh7' to guest:friends
changed ownership of `linux/redhat' retained to
guest:friends
ownership of `linux/redhat_sym' retained as guest:friends
ownership of `linux/ubuntu_sym' retained as guest:friends
changed ownership of `linux/linuxKernel' to guest:friends
changed ownership of `linux/ubuntu/ub10' to guest:friends
ownership of `linux/ubuntu' retained as guest:friends
ownership of `linux' retained as guest:friends
```

How to Copy Files in Linux and Unix? 10 cp Command Examples

thegeekstuff.com/2013/03/cp-command-examples/

cp is one of the basic command in Unix. You already know that it is used to copy one or more files or directories from source to destination.

While this tutorial is for beginners, it is also helpful for everybody to quickly review various cp command options using some practical examples.

Even if you are using cp command all the times, probably one or more examples explained below might be new to you.

The general form of copy command:

```
cp [option] source
destination
```

1. Copy a file or directory from source to destination

To copy a file, you need to pass source and destination to the copy command. The following example copies the file from project/readme.txt to projectbackup/readme-new.txt

```
$ cp project/readme.txt projectbackup/readme-
new.txt

$ cd projectbackup/

$ ls
readme-new.txt
```

If you want to copy a file from one folder to another with the same name, just the destination directory name is good enough as shown below.

```
$ cp project/readme.txt
projectbackup/

$ cd projectbackup/

$ ls
readme.txt
```

A directory (and all its content) can be copied from source to destination with the recursive option -r as shown below:

```
$ ls project
src/  bin/  doc/  lib/  test/  readme.txt
LICENSE

$ cp -r project/ backup/

$ ls backup
src/  bin/  doc/  lib/  test/  readme.txt
LICENSE
```

2. Copy multiple files or directories

You can copy more than one file from source to destination as shown below:

```
$ cd src/
$ cp global.c main.c parse.c
/home/thegeekstuff/projectbackup/src/
```

If the source files has a common pattern, use wild-cards as shown below. In this example, all c extension files gets copied to /home/thegeekstuff/projectbackup/src/ directory.

```
$ cp *.c
/home/thegeekstuff/projectbackup/src/
```

Copy multiple directories as shown below.

```
$ cd project/

$ cp -r src/ bin/
/home/thegeekstuff/projectbackup/
```

3. Backup before copying into a destination

In case if the destination file is already present with the same name, then cp allows you to backup the destination file before overwriting it.

In this example, the readme.txt exists in both project/ and projectbackup/ directory, and while copying it from project/ to projectbackup/, the existing readme.txt is backed up as shown below:

```
$ cd projectbackup

$ ls -l readme.txt
-rw-r--r-- 1 bala geek 1038 Jan  8 13:15 readme.txt

$ cd ../project

$ ls -l readme.txt
-rw-r--r-- 1 bala geek 1020 Jan  8 12:25 readme.txt

$ cp --backup readme.txt
/home/thegeekstuff/projectbackup/
```

The existing file has been moved to readme.txt~ and the new file copied as readme.txt as shown below.

```
$ cd /home/thegeekstuff/projectbackup/
$ ls -l
-rw-r--r-- 1 bala geek 1020 Jan  8 13:36 readme.txt
-rw-r--r-- 1 bala geek 1038 Jan  8 13:15
readme.txt~
```

Talking about backup, it is important for you to understand how rsync command works to backup files effectively.

4. Preserve the links while copying

When you execute the cp command, if the source is a link file, then the actual file gets copied and not the link file. In case if you only want to copy the link as it is, specify option -d as shown below:

The following shows that without option -d, it will copy the file (and not the link):

```
$ cd project/bin

$ ls -l startup.sh
lrwxrwxrwx 1 root root 18 Jan  8 13:59 startup.sh ->
../test/startup.sh

$ cp startup.sh /home/thegeekstuff/projectbackup/bin/

$ cd /home/thegeekstuff/projectbackup/bin/

$ ls -l
-rw-r--r--  1 root root       102 Jan  8 14:02 startup.sh
```

To preserve the link while copying, do the following:

```
$ cd project/bin

$ cp -d startup.sh /home/thegeekstuff/projectbackup/bin/

$ ls -l startup.sh
lrwxrwxrwx 1 root root 18 Jan  8 14:10 startup.sh ->
../test/startup.sh
```

5. Don’t overwrite an existing file

If you want to copy only when the destination file doesn’t exist, use option -n as shown below. This won’t overwrite the existing file, and cp command will return with success exit code as shown below:

```
$ cd projectbackup

$ ls -l readme.txt
-rw-r--r-- 1 bala geek 1038 Jan  8 13:15 readme.txt

$ cd ../project

$ ls -l readme.txt
-rw-r--r-- 1 bala geek 1020 Jan  8 12:25 readme.txt

$ cp -n readme.txt
/home/thegeekstuff/projectbackup/bin/

$ echo $?
0
```

As you see below, the destination file didn’t get overwritten.

```
$ cd projectbackup

$ ls -l readme.txt
-rw-r--r-- 1 bala geek 1038 Jan  8 13:15
readme.txt
```

6. Confirm before overwriting (interactive mode)

When you use -i option, it will ask for confirmation before overwriting a file as shown below.

```
$ cp -i readme.txt /home/thegeekstuff/projectbackup/
cp: overwrite `/home/thegeekstuff/projectbackup/readme.txt'?
y
```

7. Create hard link to a file (instead of copying)

When you execute cp command, it is possible to create a hard link of the file (instead of copying the file). The following example creates the hard link for sample.txt file into directory test/,

```
$ ls -li sample.txt
10883362 -rw-r--r-- 2 bala geek   1038 Jan  9 18:40 sample.txt

$ cp -l sample.txt test/

$ ls -li test/sample.txt
10883362 -rw-r--r-- 2 bala geek   1038 Jan  9 18:40
test/sample.txt
```

As seen above, the test/sample.txt is a hard linked file to sample.txt file and the inode of both files are the same.

8. Create Soft link to a file or directory (instead of copying)

When you execute cp command, it is possible to create a soft link to a file or directory. In the following example, a symbolic link gets created for libFS.so.6.0.0 as libFS.so,

```
# cd /usr/lib/

# ls -l libFS.so.6.0.0
-rw-r--r-- 1 root root 42808 Nov 19  2010 libFS.so.6.0.0

# cp -s libFS.so.6.0.0 libFS.so

# ls -l libFS.so
lrwxrwxrwx 1 root root 14 Jan  9 20:18 libFS.so ->
libFS.so.6.0.0
```

9. Preserve attributes of file or directory while copying

Using -p option, you can preserve the properties of a file or directory as shown below:

```
$ ls -l sample.txt
-rw-r--r-- 2 bala geek   1038 Jan  9 18:40 sample.txt

$ cp -p sample.txt test/

$ ls -l test/sample.txt
-rw-r--r-- 2 bala geek   1038 Jan  9 18:40
test/sample.txt
```

It is also possible to preserve only the required properties like mode, ownership, timestamps, etc.,

The following example preserves the mode of a file while copying it:

```
$ cp --preserve=mode sample.txt
test/
```

10. Copy only when source file is newer than the destination or missing

Copy doesn’t take much time for a small file, but it may take considerable amount of time when a huge file is copied. So, while copying a big file, you may want to make sure you do it only when the source file is newer than the destination file, or when the destination file is missing using the option -u as shown below.

In this example, the two files LICENSE and readme.txt will be copied from project/ to projectbackup/. However, the LICENSE file already exists in projectbackup/ directory and that is newer than the one in the project/ directory.

```
$ cd project/

$ ls -l LICENSE readme.txt
-rw-r--r-- 1 bala geek 108 Jan  8 13:14 LICENSE
-rw-r--r-- 1 bala geek 32 Jan  8 13:16 readme.txt

$ cd /home/thegeekstuff/projectbackup/

$ ls -l LICENSE readme.txt
ls: cannot access readme.txt: No such file or
directory
-rw-r--r-- 1 root root 112 Jan  9 20:31 LICENSE
```

So, in this example, there is no need to copy LICENSE file again to projectbackup/ directory. This is automatically taken care by cp command, if you use -u option as shown below. In the below example, only readme.txt file got copied as indicated by the time-stamp on the file.

```
$ cp -u -v LICENSE readme.txt
/home/thegeekstuff/projectbackup/
`readme.txt' -> `/home/thegeekstuff/projectbackup/readme.txt'

$ cd /home/thegeekstuff/projectbackup/

$ ls -l LICENSE readme.txt
-rw-r--r-- 1 bala geek  112 Jan  9 20:31 LICENSE
-rw-r--r-- 1 bala geek   32 Jan  9 22:17 readme.txt
```

Linux Crontab: 15 Awesome Cron Job Examples

thegeekstuff.com/2009/06/15-practical-crontab-examples/

An experienced Linux sysadmin knows the importance of running the routine maintenance jobs in the background automatically.
Linux Cron utility is an effective way to schedule a routine background job at a specific time and/or day on an on-going basis.
This article is part of the on-going Productivity Tips For Geeks series. In this article, let us review 15 awesome examples of crontab job scheduling.

Linux Crontab Format

```
MIN HOUR DOM MON DOW
CMD
```

Table: Crontab Fields and Allowed Ranges (Linux Crontab Syntax)

Field	Description	Allowed Value
MIN	Minute field	0 to 59
HOUR	Hour field	0 to 23
DOM	Day of Month	1-31
MON	Month field	1-12
DOW	Day Of Week	0-6
CMD	Command	Any command to be executed.

1. Scheduling a Job For a Specific Time

The basic usage of cron is to execute a job in a specific time as shown below. This will execute the Full backup shell script (full-backup) on **10th June 08:30 AM** .
Please note that the time field uses 24 hours format. So, for 8 AM use 8, and for 8 PM use 20.

```
30 08 10 06 * /home/ramesh/full-
backup
```

- **30** – 30th Minute
- **08** – 08 AM
- **10** – 10th Day
- **06** – 6th Month (June)
- ***** – Every day of the week

2. Schedule a Job For More Than One Instance (e.g. Twice a Day)

The following script take a incremental backup twice a day every day.
This example executes the specified incremental backup shell script (incremental-backup) at 11:00 and 16:00 on every day. The comma separated value in a field specifies that the command needs to be executed in all the mentioned time.

```
00 11,16 * * * /home/ramesh/bin/incremental-
backup
```

- **00** – 0th Minute (Top of the hour)
- **11,16** – 11 AM and 4 PM
- ***** – Every day
- ***** – Every month
- ***** – Every day of the week

3. Schedule a Job for Specific Range of Time (e.g. Only on Weekdays)

If you wanted a job to be scheduled for every hour with in a specific range of time then use the following.

Cron Job everyday during working hours

This example checks the status of the database everyday (including weekends) during the working hours 9 a.m – 6 p.m

```
00 09-18 * * * /home/ramesh/bin/check-db-
status
```

- **00** – 0th Minute (Top of the hour)
- **09-18** – 9 am, 10 am,11 am, 12 am, 1 pm, 2 pm, 3 pm, 4 pm, 5 pm, 6 pm
- ***** – Every day
- ***** – Every month
- ***** – Every day of the week

Cron Job every weekday during working hours

This example checks the status of the database every weekday (i.e excluding Sat and Sun) during the working hours 9 a.m – 6 p.m.

```
00 09-18 * * 1-5 /home/ramesh/bin/check-db-
status
```

- **00** – 0th Minute (Top of the hour)

- **09-18** – 9 am, 10 am,11 am, 12 am, 1 pm, 2 pm, 3 pm, 4 pm, 5 pm, 6 pm
- ***** – Every day
- ***** – Every month
- **1-5** -Mon, Tue, Wed, Thu and Fri (Every Weekday)

4. How to View Crontab Entries?

View Current Logged-In User's Crontab entries

To view your crontab entries type crontab -l from your unix account as shown below.

```
ramesh@dev-db$ crontab -l
@yearly /home/ramesh/annual-maintenance
*/10 * * * * /home/ramesh/check-disk-space

[Note: This displays crontab of the current logged in
user]
```

View Root Crontab entries

Login as root user (su – root) and do crontab -l as shown below.

```
root@dev-db# crontab -
l
no crontab for root
```

Crontab HowTo: View Other Linux User's Crontabs entries

To view crontab entries of other Linux users, login to root and use **-u {username} -l** as shown below.

```
root@dev-db# crontab -u sathiya -l
@monthly /home/sathiya/monthly-backup
00 09-18 * * * /home/sathiya/check-db-
status
```

5. How to Edit Crontab Entries?

Edit Current Logged-In User's Crontab entries

To edit a crontab entries, use crontab -e as shown below. By default this will edit the current logged-in users crontab.

```
ramesh@dev-db$ crontab -e
@yearly /home/ramesh/centos/bin/annual-maintenance
*/10 * * * * /home/ramesh/debian/bin/check-disk-space
~
"/tmp/crontab.XXXXyjWkHw" 2L, 83C

[Note: This will open the crontab file in Vim editor for
editing.
Please note cron created a temporary /tmp/crontab.XX... ]
```

When you save the above temporary file with :wq, it will save the crontab and display the following message indicating the crontab is successfully modified.

```
~
"crontab.XXXXyjWkHw" 2L, 83C
written
crontab: installing new crontab
```

Edit Root Crontab entries

Login as root user (su – root) and do crontab -e as shown below.

```
root@dev-db# crontab -
e
```

Edit Other Linux User's Crontab File entries

To edit crontab entries of other Linux users, login to root and use **-u {username} -e** as shown below.

```
root@dev-db# crontab -u sathiya -e
@monthly /home/sathiya/fedora/bin/monthly-backup
00 09-18 * * * /home/sathiya/ubuntu/bin/check-db-
status
~
~
~
"/tmp/crontab.XXXXyjWkHw" 2L, 83C
```

6. Schedule a Job for Every Minute Using Cron.

Ideally you may not have a requirement to schedule a job every minute. But understanding this example will will help you understand the other examples mentioned below in this article.

```
* * * * *
CMD
```

The * means all the possible unit — i.e every minute of every hour through out the year. More than using this * directly, you will find it very useful in the following cases.

- When you specify */5 in minute field means every 5 minutes.
- When you specify 0-10/2 in minute field mean every 2 minutes in the first 10 minute.
- Thus the above convention can be used for all the other 4 fields.

7. Schedule a Background Cron Job For Every 10 Minutes.

Use the following, if you want to check the disk space every 10 minutes.

```
*/10 * * * * /home/ramesh/check-disk-
space
```

It executes the specified command check-disk-space every 10 minutes through out the year. But you may have a requirement of executing the command only during office hours or vice versa. The above examples shows how to do those things.
Instead of specifying values in the 5 fields, we can specify it using a single keyword as mentioned below.
There are special cases in which instead of the above 5 fields you can use @ followed by a keyword — such as reboot, midnight, yearly, hourly.

Table: Cron special keywords and its meaning

Keyword	Equivalent
@yearly	0 0 1 1 *
@daily	0 0 * * *
@hourly	0 * * * *
@reboot	Run at startup.

8. Schedule a Job For First Minute of Every Year using @yearly

If you want a job to be executed on the first minute of every year, then you can use the **@yearly** cron keyword as shown below.
This will execute the system annual maintenance using annual-maintenance shell script at 00:00 on Jan 1st for every year.

```
@yearly /home/ramesh/red-hat/bin/annual-maintenance
```

9. Schedule a Cron Job Beginning of Every Month using @monthly

It is as similar as the @yearly as above. But executes the command monthly once using **@monthly** cron keyword. This will execute the shell script tape-backup at 00:00 on 1st of every month.

```
@monthly /home/ramesh/suse/bin/tape-
backup
```

10. Schedule a Background Job Every Day using @daily

Using the **@daily** cron keyword, this will do a daily log file cleanup using cleanup-logs shell scriptat 00:00 on every day.

```
@daily /home/ramesh/arch-linux/bin/cleanup-logs "day
started"
```

11. How to Execute a Linux Command After Every Reboot using @reboot?

Using the **@reboot** cron keyword, this will execute the specified command once after the machine got booted every time.

```
@reboot
CMD
```

12. How to Disable/Redirect the Crontab Mail Output using MAIL keyword?

By default crontab sends the job output to the user who scheduled the job. If you want to redirect the output to a specific user, add or update the MAIL variable in the crontab as shown below.

```
ramesh@dev-db$ crontab -l
MAIL="ramesh"

@yearly /home/ramesh/annual-maintenance
*/10 * * * * /home/ramesh/check-disk-space

[Note: Crontab of the current logged in user with MAIL
variable]
```

If you wanted the mail not to be sent to anywhere, i.e to stop the crontab output to be emailed, add or update the MAIL variable in the crontab as shown below.

```
MAIL=""
```

13. How to Execute a Linux Cron Jobs Every Second Using Crontab.

You cannot schedule a every-second cronjob. Because in cron the minimum unit you can specify is minute. In a typical scenario, there is no reason for most of us to run any job every second in the system.

14. Specify PATH Variable in the Crontab

All the above examples we specified absolute path of the Linux command or the shell-script that needs to be executed.
For example, instead of specifying /home/ramesh/tape-backup, if you want to just specify tape-backup, then add the path /home/ramesh to the PATH variable in the crontab as shown below.

```
ramesh@dev-db$ crontab -l

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/home/ramesh

@yearly annual-maintenance
*/10 * * * * check-disk-space

[Note: Crontab of the current logged in user with PATH
variable]
```

15. Installing Crontab From a Cron File

Instead of directly editing the crontab file, you can also add all the entries to a cron-file first. Once you have all thoese entries in the file, you can upload or install them to the cron as shown below.

```
ramesh@dev-db$ crontab -l
no crontab for ramesh

$ cat cron-file.txt
@yearly /home/ramesh/annual-maintenance
*/10 * * * * /home/ramesh/check-disk-
space

ramesh@dev-db$ crontab cron-file.txt

ramesh@dev-db$ crontab -l
@yearly /home/ramesh/annual-maintenance
*/10 * * * * /home/ramesh/check-disk-
space
```

Note: This will install the cron-file.txt to your crontab, which will also remove your old cron entries. So, please be careful while uploading cron entries from a cron-file.txt.

Additional Cron Tutorials

Awesome Linux Articles

Following are few awesome **15 examples** articles that you might find helpful.

mtime, ctime, and atime

mtime, ctime, and atime

Unix keeps 3 timestamps for each file: mtime, ctime, and atime. Most people seem to understand atime (access time), it is when the file was last read. There does seem to be some confusion between mtime and ctime though. ctime is the inode change time while mtime is the file modification time. "Change" and "modification" are pretty much synonymous. There is no clue to be had by pondering those words. Instead you need to focus on what is being changed. mtime changes when you write to the file. It is the age of the data in the file. Whenever mtime changes, so does ctime. But ctime changes a few extra times. For example, it will change if you change the owner or the permissions on the file.

Let's look at a concrete example. We run a package called Samba that lets PC's access files. To change the Samba configuration, I just edit a file called smb.conf. (This changes mtime and ctime.) I don't need to take any other action to tell Samba that I changed that file. Every now and then Samba looks at the mtime on the file. If the mtime has changed, Samba rereads the file. Later that night our backup system runs. It uses ctime, which also changed so it backs up the file. But let's say that a couple of days later I notice that the permissions on smb.conf are 666. That's not good..anyone can edit the file. So I do a "chmod 644 smb.conf". This changes only ctime. Samba will not reread the file. But later that night, our backup program notices that ctime has changes, so it backs up the file. That way, if we lose the system and need to reload our backups, we get the new improved permission setting.

Here is a second example. Let's say that you have a data file called employees.txt which is a list of employees. And you have a program to print it out. The program not only prints the data, but it obtains the mtime and prints that too. Now someone has requested an employee list from the end of the year 2000 and you found a backup tape that has that file. Many restore programs will restore the mtime as well. When you run that program it will print an mtime from the end of the year 2000. But the ctime is today. So again, our backup program will see the file as needing to be backed up.

Suppose your restore program did not restore the mtime. You don't want your program to print today's date. Well no problem. mtime is under your control. You can set it to what ever you want. So just do:

Code:

```
$ touch -t 200012311800
employees.txt
```

This will set mtime back to the date you want and it sets ctime to now. You have complete control over mtime, but the system stays in control of ctime. So mtime is a little bit like the date on a letter while ctime is like the postmark on the envelope.

find command -mtime -ctime -atime

The find command uses arguments like:
-mtime -2
-mtime +2
-mtime 2

There are -ctime and -atime options as well. Since we now understand the differences among mtime, ctime, and atime, by understanding how find uses the -mtime option, the other two become understood as well. So I will describe find's use of the -mtime option.

As you probably know, the find command can run for minutes or hours depending on the size of the filesystem being searched. The find command makes a note of its own start time. It then looks at a file's mtime and computes how many seconds ago the file was modified. By dividing the seconds by 86,400 (and discarding any remainder), it can calculate the file's age in days:

Code:

```
0 days in seconds:       0  -
86399
1 day in seconds:    86400  -
172799
2 days in seconds:  172800  -
259159
```

So now that we know how many days ago a file was modified, we can use stuff like "-mtime 2" which specifies files that are 172800 to 259159 seconds older than the instant that the find command was started.

"-mtime -2" means files that are less than 2 days old, such as a file that is 0 or 1 days old.

"-mtime +2" means files that are more than 2 days old... {3, 4, 5, ...}

It may seem odd, but +0 is supposed to work and would mean files more than 0 days old. It is very important to recognize that find's concept of a "day" has nothing to do with midnight.

Last edited by Perderabo; 08-05-2007 at 11:40 AM..

Using perl to display the file timestamps

The ls program will display mtime if you use "ls -l". And you can get atime or ctime with "ls -lu" or "ls -lc". But ls uses a strange format. It displays the month and day in all cases. If the timestamp is recent, it also displays hour and minute. If the timestamp is older than 6 months, it display the year instead of hour and minute. A clever script can reformat this to year, month, day, hour, and minute. But ls will not display the seconds. The gnu version of ls (which is usually the only version on linux) does have extended options like --fulltime. But these extended options are non-standard and won't be available on other versions of Unix.

The perl language is also non-standard, but perl tends to be available on most versions of unix. For example, a version of perl is supplied with HP-UX and Solaris. Perl can easily display the timestamps of files. Here are some perl one-liners to display atime, mtime, and ctime.

Code:

```
$ echo hello > testfile ; date
Thu Aug 30 08:31:57 EDT 2007
$ chmod 700 testfile ; date
Thu Aug 30 08:32:48 EDT 2007
$ cat testfile ; date
hello
Thu Aug 30 08:33:30 EDT 2007
$
$
$
$
$ perl -e '@d=localtime ((stat(shift))[8]); printf "%4d%02d%02d%02d%02d%02d\n",
$d[5]+1900,$d[4]+1,$d[3],$d[2],$d[1],$d[0]' testfile
20070830083330
$ perl -e '@d=localtime ((stat(shift))[9]); printf "%4d%02d%02d%02d%02d%02d\n",
$d[5]+1900,$d[4]+1,$d[3],$d[2],$d[1],$d[0]' testfile
20070830083157
$ perl -e '@d=localtime ((stat(shift))[10]); printf "%4d%02d%02d%02d%02d%02d\n",
$d[5]+1900,$d[4]+1,$d[3],$d[2],$d[1],$d[0]' testfile
20070830083248
$
```

How to use double or single brackets, parentheses, curly braces

stackoverflow.com/questions/2188199/how-to-use-double-or-single-brackets-parentheses-curly-braces/5

6 Answers

In Bash, `test` and `[` are builtins.

The double bracket enables additional functionality. For example, you can use `&&` and `||` instead of `-a` and `-o` and there's a regular expression matching operator `=~`.

The braces, in addition to delimiting a variable name are used for parameter expansion so you can do things like:

- Truncate the contents of a variable

  ```
  $ var="abcde"; echo
  ${var%d*}
  abc
  ```

- Make substitutions similar to `sed`

  ```
  $ var="abcde"; echo
  ${var/de/12}
  abc12
  ```

- Use a default value

  ```
  $ default="hello"; unset var; echo ${var:-
  $default}
  hello
  ```

- and several more

Also, brace expansions create lists of strings which are typically iterated over in loops:

```
$ echo f{oo,ee,a}d
food feed fad

$ mv error.log{,.OLD}
(error.log is renamed to error.log.OLD because the brace
expression
expands to "mv error.log error.log.OLD")

$ for num in {000..2}; do echo "$num"; done
000
001
002

$ echo {00..8..2}
00 02 04 06 08

$ echo {D..T..4}
D H L P T
```

Note that the leading zero and increment features weren't available before Bash 4.

Thanks to gboffi for reminding me about brace expansions.

Double parentheses are used for arithmetic operations:

```
((a++))

((meaning = 42))

for ((i=0; i<10; i++))

echo $((a + b + (14 *
c)))
```

and they enable you to omit the dollar signs on integer and array variables and include spaces around operators for readability.

Single brackets are also used for array indices:

```
array[4]="hello"

element=${array[index]}
```

Curly brace are required for (most/all?) array references on the right hand side.

ephemient's comment reminded me that parentheses are also used for subshells. And that they are used to create arrays.

```
array=(1 2 3)
echo
${array[1]}
2
```

1. A single bracket (`[`) usually actually calls a program named `[`; `man test` or `man [` for more info. Example:

```
$ VARIABLE=abcdef
$ if [ $VARIABLE == abcdef ] ; then echo yes ; else echo no ;
fi
yes
```

2. The double bracket (`[[`) does the same thing (basically) as a single bracket, but is a bash builtin.

```
$ VARIABLE=abcdef
$ if [[ $VARIABLE == 123456 ]] ; then echo yes ; else echo no ;
fi
no
```

3. Parentheses (`()`) are used to create a subshell. For example:

```
$ pwd
/home/user
$ (cd /tmp;
pwd)
/tmp
$ pwd
/home/user
```

   As you can see, the subshell allowed you to perform operations without affecting the environment of the current shell.

4a. Braces (`{}`) are used to unambiguously identify variables. Example:

```
    $ VARIABLE=abcdef
    $ echo Variable: $VARIABLE
    Variable: abcdef
    $ echo Variable: $VARIABLE123456
    Variable:
    $ echo Variable:
${VARIABLE}123456
    Variable: abcdef123456
```

4b. Braces are also used to execute a sequence of commands in the *current* shell context, e.g.

```
    $ { date; top -b -n1 | head ; } >logfile
    # 'date' and 'top' output are concatenated,
    # could be useful sometimes to hunt for a top loader )

    $ { date; make 2>&1; date; } | tee logfile
    # now we can calculate the duration of a build from the
logfile
```

There is a subtle syntactic difference with `( )`, though (see bash reference) ; essentially, a semicolon `;` after the last command within braces is a must, and the braces `{`, `}` **must** be surrounded by spaces.

Brackets

```
if [ CONDITION ]    Test construct
if [[ CONDITION ]]  Extended test construct
Array[1]=element1   Array initialization
[a-z]               Range of characters within a Regular Expression
$[ expression ]     A non-standard & obsolete version of $(( expression ))
[1]
```

[1] http://wiki.bash-hackers.org/scripting/obsolete

Curly Braces

```
${variable}                             Parameter substitution
${!variable}                            Indirect variable reference
{ command1; command2; . . . commandN; } Block of code
{string1,string2,string3,...}           Brace expansion
{a..z}                                  Extended brace expansion
{}                                      Text replacement, after find and
xargs
```

Parentheses

```
( command1; command2 )             Command group executed within a subshell
Array=(element1 element2 element3) Array initialization
result=$(COMMAND)                  Command substitution, new style
>(COMMAND)                         Process substitution
<(COMMAND)                         Process substitution
```

Double Parentheses

```
(( var = 78 ))            Integer arithmetic
var=$(( 20 + 5 ))         Integer arithmetic, with variable assignment
(( var++ ))               C-style variable increment
(( var-- ))               C-style variable decrement
(( var0 = var1<98?9:21 )) C-style ternary operation
```

I just wanted to add these from TLDP:

```
~:$ echo $SHELL
/bin/bash

~:$ echo ${#SHELL}
9

~:$ ARRAY=(one two three)

~:$ echo ${#ARRAY}
3
```

```
~:$ echo ${TEST:-test}
test

~:$ echo $TEST


~:$ export TEST=a_string

~:$ echo ${TEST:-test}
a_string

~:$ echo ${TEST2:-$TEST}
a_string

~:$ echo $TEST2


~:$ echo ${TEST2:=$TEST}
a_string

~:$ echo $TEST2
a_string

~:$ export
STRING="thisisaverylongname"

~:$ echo ${STRING:4}
isaverylongname

~:$ echo ${STRING:6:5}
avery

~:$ echo ${ARRAY[*]}
one two one three one four

~:$ echo ${ARRAY[*]#one}
two three four

~:$ echo ${ARRAY[*]#t}
one wo one hree one four

~:$ echo ${ARRAY[*]#t*}
one wo one hree one four

~:$ echo ${ARRAY[*]##t*}
one one one four

~:$ echo $STRING
thisisaverylongname

~:$ echo ${STRING%name}
thisisaverylong

~:$ echo ${STRING/name/string}
thisisaverylongstring
```

The difference between **test**, **[** and **[[** is explained in great details in the BashFAQ.

> *To cut a long story short: test implements the old, portable syntax of the command. In almost all shells (the oldest Bourne shells are the exception), [is a synonym for test (but requires a final argument of]). Although all modern shells have built-in implementations of [, there usually still is an external executable of that name, e.g. /bin/[.*
>
> *[[is a new improved version of it, which is a keyword, not a program. This has beneficial effects on the ease of use, as shown below. [[is understood by KornShell and BASH (e.g. 2.03), but not by the older POSIX or BourneShell.*

And the conclusion:

> *When should the new test command [[be used, and when the old one [? If portability to the BourneShell is a concern, the old syntax should be used. If on the other hand the script requires BASH or KornShell, the new syntax is much more flexible.*

Parentheses in function definition

Parentheses `()` are being used in function definition:

```
function_name () { command1 ; command2 ;
}
```

That is the reason you have to escape parentheses even in command parameters:

```
$ echo (
bash: syntax error near unexpected token `newline'

$ echo \(
(

$ echo () { command echo The command echo was redefined. ;
}
$ echo anything
The command echo was redefined.
```

7 Linux Grep OR, Grep AND, Grep NOT Operator Examples

thegeekstuff.com/2011/10/grep-or-and-not-operators

10/21/2011

by Ramesh Natarajan on October 21, 2011

Question: Can you explain how to use OR, AND and NOT operators in Unix grep command with some examples?

Answer: In grep, we have options equivalent to OR and NOT operators. There is no grep AND opearator. But, you can simulate AND using patterns. The examples mentioned below will help you to understand how to use OR, AND and NOT in Linux grep command.

The following employee.txt file is used in the following examples.

```
$ cat employee.txt
100  Thomas  Manager    Sales       $5,000
200  Jason   Developer  Technology  $5,500
300  Raj     Sysadmin   Technology  $7,000
400  Nisha   Manager    Marketing   $9,500
500  Randy   Manager    Sales       $6,000
```

You already knew that grep is extremely powerful based on these grep command examples.

Grep OR Operator

Use any one of the following 4 methods for grep OR. I prefer method number 3 mentioned below for grep OR operator.

1. Grep OR Using \|

If you use the grep command without any option, you need to use \| to separate multiple patterns for the or condition.

```
grep 'pattern1\|pattern2' filename
```

For example, grep either Tech or Sales from the employee.txt file. Without the back slash in front of the pipe, the following will not work.

```
$ grep 'Tech\|Sales' employee.txt
100  Thomas  Manager    Sales       $5,000
200  Jason   Developer  Technology  $5,500
300  Raj     Sysadmin   Technology  $7,000
500  Randy   Manager    Sales       $6,000
```

2. Grep OR Using -E

grep -E option is for extended regexp. If you use the grep command with -E option, you just need to use | to separate multiple patterns for the or condition.

```
grep -E 'pattern1|pattern2' filename
```

For example, grep either Tech or Sales from the employee.txt file. Just use the | to separate multiple OR patterns.

```
$ grep -E 'Tech|Sales' employee.txt
100  Thomas  Manager    Sales       $5,000
200  Jason   Developer  Technology  $5,500
300  Raj     Sysadmin   Technology  $7,000
500  Randy   Manager    Sales       $6,000
```

3. Grep OR Using egrep

egrep is exactly same as 'grep -E'. So, use egrep (without any option) and separate multiple patterns for the or condition.

```
egrep 'pattern1|pattern2' filename
```

For example, grep either Tech or Sales from the employee.txt file. Just use the | to separate multiple OR patterns.

```
$ egrep 'Tech|Sales' employee.txt
100  Thomas  Manager    Sales       $5,000
200  Jason   Developer  Technology  $5,500
300  Raj     Sysadmin   Technology  $7,000
500  Randy   Manager    Sales       $6,000
```

4. Grep OR Using grep -e

Using grep -e option you can pass only one parameter. Use multiple -e option in a single command to use multiple patterns for the or condition.

```
grep -e pattern1 -e pattern2 filename
```

For example, grep either Tech or Sales from the employee.txt file. Use multiple -e option with grep for the multiple OR patterns.

```
$ grep -e Tech -e Sales employee.txt
100  Thomas  Manager    Sales       $5,000
200  Jason   Developer  Technology  $5,500
300  Raj     Sysadmin   Technology  $7,000
500  Randy   Manager    Sales       $6,000
```

Grep AND

5. Grep AND using -E 'pattern1.*pattern2'

There is no AND operator in grep. But, you can simulate AND using grep -E option.

```
grep -E 'pattern1.*pattern2' filename
grep -E 'pattern1.*pattern2|pattern2.*pattern1' filename
```

The following example will grep all the lines that contain both "Dev" and "Tech" in it (in the same order).

```
$ grep -E 'Dev.*Tech' employee.txt
200  Jason   Developer  Technology  $5,500
```

The following example will grep all the lines that contain both "Manager" and "Sales" in it (in any order).

```
$ grep -E 'Manager.*Sales|Sales.*Manager' employee.txt
```

Note: Using regular expressions in grep is very powerful if you know how to use it effectively.

6. Grep AND using Multiple grep command

You can also use multiple grep command separated by pipe to simulate AND scenario.

```
grep -E 'pattern1' filename | grep -E 'pattern2'
```

The following example will grep all the lines that contain both “Manager” and “Sales” in the same line.

```
$ grep Manager employee.txt | grep Sales
100  Thomas  Manager    Sales       $5,000
500  Randy   Manager    Sales       $6,000
```

Grep NOT

7. Grep NOT using grep -v

Using grep -v you can simulate the NOT conditions. -v option is for invert match. i.e It matches all the lines except the given pattern.

```
grep -v 'pattern1' filename
```

For example, display all the lines except those that contains the keyword “Sales”.

```
$ grep -v Sales employee.txt
200  Jason   Developer  Technology  $5,500
300  Raj     Sysadmin   Technology  $7,000
400  Nisha   Manager    Marketing   $9,500
```

You can also combine NOT with other operator to get some powerful combinations.

For example, the following will display either Manager or Developer (bot ignore Sales).

```
$ egrep 'Manager|Developer' employee.txt | grep -v Sales
200  Jason   Developer  Technology  $5,500
400  Nisha   Manager    Marketing   $9,500
```

> Add your comment

If you enjoyed this article, you might also like..

Linux and Unix ln command

About ln

ln creates links between files.

Description

ln creates a link to file *TARGET* with the name *LINKNAME*. If *LINKNAME* is omitted, a link to *TARGET* is created in the current directory, using the name of *TARGET* as the *LINKNAME*.

ln creates hard links by default, or symbolic links if the **-s** (**--symbolic**) option is specified. When creating hard links, each *TARGET* must exist.

What Is A Link?

Before we discuss the **ln** command, let's first discuss the **link** command, as well as what a link is and how it relates to files as we know them.

A *link* is an entry in your file system which connects a filename to the actual bytes of data on the disk. More than one filename can "link" to the same data. Here's an example. Let's create a file named **file1.txt**:

```
echo "This is a file." >
file1.txt
```

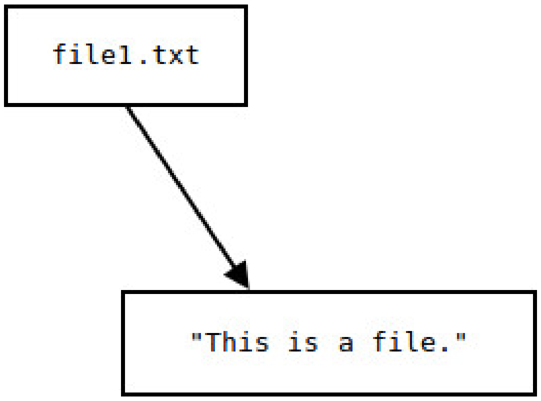
This command **echo**es the string "**This is a file**". Normally this would simply echo to our terminal, but the **>** operator redirects the string's text to a file, in this case **file1.txt**. We can check that it worked by using **cat** to display the contents of the file:

```
cat           This is a
file1.txt     file.
```

When this file was created, the operating system wrote the bytes to a location on the disk and also *linked* that data to a filename, **file1.txt** so that we can refer to the file in commands and arguments. If you rename the file, the contents of the file are not altered; only the information that *points* to it. *The filename and the file's data are two separate entities.*

Here's an illustration of the filename and the data to help you visualize it:

Using The link Command



What the **link** command does is allow us to manually create a link to file data that already exists. So, let's use **link** to create our own link to the file data we just created. In essence, we'll create another file name for the data that already exists.

Let's call our new link **file2.txt**. How do we create it?

The general form of the **link** command is: "**link** *filename linkname*". Our first argument is the name of the file whose data we're linking to; the second argument is the name of the new link we're creating.

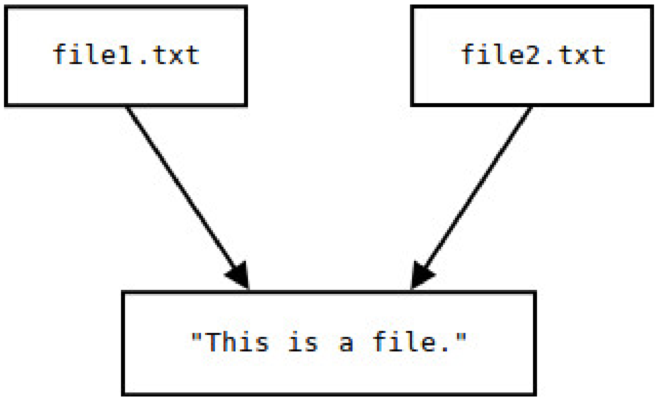
```
link file1.txt file2.txt
```

Now both **file1.txt** and **file2.txt** point to the same data on the disk:

```
cat          This is a      cat          This is a
file1.txt    file.          file2.txt    file.
```

The important thing to realize is that we did *not make a copy* of this data. Both filenames point to the same bytes of data on the disk. Here's an illustration to help you visualize it:

If we change the contents of the data pointed to by either one of these files, the other file's contents are changed as well. Let's append a line to one of them using the **>>** operator:



```
echo "It points to data on the disk." >>
file1.txt
```

Now let's look at the contents of **file1.txt**:

```
             This is a file.
cat          It points to data on the
file1.txt    disk.
```

... and now let's look at the second file, the one we created with the **link** command:

```
              This is a file.
cat           It points to data on the
file2.txt     disk.
```

Both files show the change because they share the same data on the disk. Changes to the data of either one of these files will change the contents of the other.

But what if we delete one of the files? Will both files be deleted?

No. If we delete one of the files, we're simply deleting one of the links to the data. Because we created another link manually, we still have a pointer to that data; we still have a way, at the user-level, to access the data we put in there. So if we use the **rm** command to remove our first file:

```
rm file1.txt
```

...it no longer exists as a file with that name:

```
cat           cat: file1.txt: No such file or
file1.txt     directory
```

...but the link to the data we manually created still exists, and still points to the data:

```
              This is a file.
cat           It points to data on the
file2.txt     disk.
```

As you can see, the data stays on the disk even after the "file" (which is actually just a link to the data) is removed. We can still access that data as long as there is a link to it. This is important to know when you're removing files — "removing" a file just makes the data inaccessible by **unlink**-ing it. The data still exists on the storage media, somewhere, inaccessible to the system, and that space on disk is marked as being available for future use.

The type of link we've been working with here is sometimes called a "hard" link. A hard link and the data it links to must always exist on the same filesystem; you can't, for instance, create a hard link on one partition to file data stored on another partition. You also can't create a hard link to a directory. Only symbolic links may link to a directory; we'll get to that in a moment.

The Difference Between ln And link

So what about **ln**? That's why we're here, right?

ln, by default, creates a hard link just like **link** does. So this **ln** command:

```
ln file1.txt file2.txt
```

...is the same as the following **link** command:

```
link file1.txt file2.txt
```

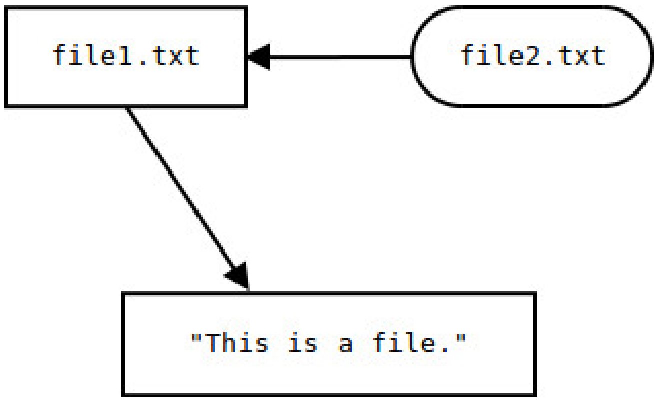
...because both commands create a hard link named **file2.txt** which links to the data of **file1.txt**.

However, we can also use **ln** to create *symbolic links* with the **-s** option. So the command:

```
ln -s file1.txt
file2.txt
```

Will create a symbolic link to **file1.txt** named **file2.txt**. In contrast to our hard link example, here's an illustration to help you visualize our symbolic link:

About Symbolic Links



Symbolic links, sometimes called "soft" links, are different than "hard" links. Instead of linking to the data of a file, they *link to another link*. So in the example above, **file2.txt** points to the link **file1.txt**, which in turn points to the data of the file.

This has several potential benefits. For one thing, symbolic links (also called "symlinks" for short) can link to directories. Also, symbolic links can cross file system boundaries, so a symbolic link to data on one drive or partition can exist on another drive or partition.

You should also be aware that, unlike hard links, removing the file (or directory) that a symlink points to will break the link. So if we create **file1.txt**:

```
echo "This is a file." >
file1.txt
```

...and create a symbolic link to it:

```
ln -s file1.txt
file2.txt
```

...we can **cat** either one of these to see the contents:

```
cat          This is a      cat          This is a
file1.txt    file.          file2.txt    file.
```

...but if we remove **file1.txt**:

```
rm file1.txt
```

...we can no longer access the data it contained with our symlink:

```
cat           cat: file2.txt: No such file or
file2.txt     directory
```

This error message might be confusing at first, because **file2.txt** still exists in your directory. It's a broken symlink, however — a symbolic link which points to something that no longer exists. The operating system tries to follow the symlink to the file that's supposed to be there (**file1.txt**), but finds nothing, and so it returns the error message.

While hard links are an essential component of how the operating system works, symbolic links are generally more of a convenience. You can use them to refer, in any way you'd like, to information already on the disk somewhere else.

Creating Symlinks To Directories

To create a symbolic link to a directory, simply specify the directory name as the target. For instance, let's say we have a directory named **documents**, which contains one file, named **file.txt**.

Let's create a symbolic link to **documents** named **dox**. This command will do the trick:

```
ln -s documents/
dox
```

We now have a symlink named **dox** which we can refer to *as if it is the directory **documents***. For instance, if we use **ls** to list the contents of the directory, and then to list the contents of the symlinked directory, they will both show the same file:

```
                        ls
ls documents file.txt dox    file.txt
```

When we work in the directory **dox** now, we will actually be working in **documents**, but we will see the word **dox** instead of **documents** in all pathnames.

Symbolic links are a useful way to make shortcuts to long, complicated pathnames. For instance, this command:

```
ln -s documents/work/budgets/Engineering/2014/April
aprbudge
```

...will save us a lot of typing; now, instead of changing directory with the following command:

```
cd documents/work/budgets/Engineering/2014/April
```

...we can do this, instead:

```
cd
aprbudge
```

Normally, you remove directories (once they're empty) with the **rmdir** command. But our symbolic link is not actually a directory: it's a file that points to a directory. So to remove our symlink, we just use the **rm** command:

```
rm
aprbudge
```

This will remove the symlink, but the original directory and all its files are not affected.

ln syntax

```
ln [OPTION]... TARGET [...] [LINKNAME
[...]]
```

Options

Here are the options that can be passed to the **ln** command.

--backup[=*CONTROL*]	Use this option to additionally create a backup of each existing destination file. The style of backup is optionally defined by the value of *CONTROL*. See below for more information.
-b	This functions like **--backup**, but you cannot specify the *CONTROL*; the default style (**simple**) is used.
-d, **-F**, **--directory**	This option allows the superuser to attempt to hard link directories (although it will probably fail due to system restrictions, even for the superuser).
-f, **--force**	If the destination file or files already exist, overwrite them.
-i, **--interactive**	Prompt the user before overwriting destination files.
-L, **--logical**	Dereference *TARGET*s that are symbolic links. In other words, if you are trying to create a link (or a symlink) to a symlink, link to what it links to, not to the symlink itself..
-n, **--no-dereference**	Treat *LINKNAME* as a normal file if it is a symbolic link to a directory.
-P, **--physical**	Make hard links directly to symbolic links, rather than dereferencing them.
-r, **--relative**	Create symbolic links relative to link location.
-s, **--symbolic**	Make symbolic links instead of hard links.
-S, **--suffix=***SUFFIX*	Use the file suffix *SUFFIX* rather than the default suffix "**~**".
-t, **--target-directory=***DIRECTORY*	Specify the *DIRECTORY* in which to create the links.
-T, **--no-target-directory**	Always treat *LINKNAME* as a normal file.
-v, **--verbose**	Operate verbosely; print the name of each linked file.
--help	Display a help message, and exit.
--version	Display version information, and exit.

About The --backup Option

When using the **--backup** (or **-b**) option, the default file suffix for backups is '**~**'. You can change this, however, using the **--suffix** option or setting the **SIMPLE_BACKUP_SUFFIX** environment variable.

The *CONTROL* argument to the **--backup** option specifies the version control method. Alternatively, it can be specified by setting the **VERSION_CONTROL** environment variable. Here are the values to use for either one:

none, **off**	never make backups (even if **--backup** is given)
numbered, **t**	make numbered backups.
existing, **nil**	numbered if numbered backups exist, simple otherwise.
simple, **never**	always make simple backups.

If you use **-b** instead of **--backup**, the *CONTROL* method is always *simple*.

If you specify the **-s** option (which symlinks), **ln** ignores the **-L** and **-P** options. Otherwise (if you are making hard links), the last option specified controls behavior when a *TARGET* is a symbolic link. The default is to act as if **-P** was specified.

ln examples

```
ln public_html/myfile1.txt
```

Create a hard link to the file **public_html/myfile1.txt** in the current directory.

```
ln -s
public_html/myfile1.txt
```

Create a symbolic link to the file **public_html/myfile1.txt** in the current directory.

```
ln -s public_html/
webstuff
```

Create a symbolic link to the directory **public_html** named **webstuff**.

```
ln -s -b file1.txt
file2.txt
```

Creates a symbolic link to the file **file1.txt** named **file2.txt**. If **file2.txt** already exists, it is renamed to **file2.txt~** before the new **file2.txt** symlink is created.

Example syntax for Secure Copy (scp)

What is Secure Copy?

scp allows files to be copied to, from, or between different hosts. It uses **ssh** for data transfer and provides the same authentication and same level of security as **ssh**.

Examples

Copy the file "foobar.txt" from a remote host to the local host

```
$ scp your_username@remotehost.edu:foobar.txt /some/local/directory
```

Copy the file "foobar.txt" from the local host to a remote host

```
$ scp foobar.txt your_username@remotehost.edu:/some/remote/directory
```

Copy the directory "foo" from the local host to a remote host's directory "bar"

```
$ scp -r foo your_username@remotehost.edu:/some/remote/directory/bar
```

Copy the file "foobar.txt" from remote host "rh1.edu" to remote host "rh2.edu"

```
$ scp your_username@rh1.edu:/some/remote/directory/foobar.txt \
your_username@rh2.edu:/some/remote/directory/
```

Copying the files "foo.txt" and "bar.txt" from the local host to your home directory on the remote host

```
$ scp foo.txt bar.txt your_username@remotehost.edu:~
```

Copy the file "foobar.txt" from the local host to a remote host using port 2264

```
$ scp -P 2264 foobar.txt your_username@remotehost.edu:/some/remote/directory
```

Copy multiple files from the remote host to your current directory on the local host

```
$ scp your_username@remotehost.edu:/some/remote/directory/\{a,b,c\} .
```

```
$ scp your_username@remotehost.edu:~/\{foo.txt,bar.txt\} .
```

scp Performance

By default **scp** uses the Triple-DES cipher to encrypt the data being sent. Using the Blowfish cipher has been shown to increase speed. This can be done by using option *-c blowfish* in the command line.

```
$ scp -c blowfish some_file your_username@remotehost.edu:~
```

It is often suggested that the *-C* option for compression should also be used to increase speed. The effect of compression, however, will only significantly increase speed if your connection is very slow. Otherwise it may just be adding extra burden to the CPU. An example of using blowfish and compression:

```
$ scp -c blowfish -C local_file your_username@remotehost.edu:~
```

10 Examples of tar command in UNIX and Linux

javarevisited.blogspot.in/2011/11/tar-command-in-unix-linux-example.html

tar command in UNIX or Linux is one of the important command which provides archiving functionality in unix. we can use UNIX tar command to create compressed or uncompressed archive files by using either gzip or bzip2. In this **unix tar command tutorial** we will see examples of unix tar command related to basic archiving task e.g. **How to create tar archive in Unix and Linux**, How to extract files from tar archive in unix, How to view contents of tar file in Unix and Linux or how to update and existing tar file in Unix. Examples of tar command in unix are kept simple and easy to understand and master each of basic task using **unix tar command**.

I thought about this article when I written how to be productive in UNIX and UNIX command tutorial and Example for beginners but somehow it gets delayed and now I am happy to see this published.
Ok enough introduction now let's see some *real life examples of tar command in Unix and Linux*:

How to use tar command in Unix

Using tar command in UNIX is simple and it has similar syntax like any other UNIX command. below is the syntax of tar command in UNIX:

tar [options] [name of tar file to be created] [list of files and directories to be included]

This **syntax of tar command** is for easy understanding you can also check detailed syntax by using command "tar --usage" in UNIX machine.

tar command examples in Linux

Unix tar command line options

In this section of UNIX tar command tutorial, we will see some useful options of tar command in Linux and we will use this options in our example to understand the usage of this option along-with tar command.

c -- create, for creating tar file
v -- verbose, the display name of files including,excluding from tar command
f -- following, used to point name of tar file to be created. it actually tells tar command that name of the file is "next" letter just after options.

x -- extract, for extracting files from the tar file.
t -- for viewing the content of tar file
z -- zip, tells tar command that creates tar file using gzip.
j –- another compressing option tells tar command to use bzip2 for compression
r -- update or adds file or directory in already existed .tar file
wildcards -- to specify patters in Unix tar command

How to create tar archive or tar file in Unix

Most of the use either WinZip or WinRAR in windows machine to zipping or creating archives of content so when we move to command line interface like Unix or Linux we struggle without those tools. UNIX tar command is

similar to WinZip or WinRAR and you can use UNIX tar command to create both compressed or uncompressed (zipped) archives in UNIX.

In this example of tar command, we will create tar file including all the files and directories or selected files and directories in Unix.

here is our directory

```
stock_trader@system:~/test ls -lrt
total 0
-r--r--r--  1 stock_trader Domain Users 0 Jul 15 11:42 equity
drwxrwxrwx+ 1 stock_trader Domain Users 0 Jul 15 14:33 stocks/
-r--r--r--  1 stock_trader Domain Users 0 Jul 15 15:30 currency
```

it has two files and one directory. now we will create a tar file with all these contents.

```
stock_trader@system:~/test tar -cvf trading.tar *
currency
equity
stocks/
stocks/online_stock_exchanges.txt
```

You see unix tar command is creating tar file with name "**trading**" with contents shown above. just to review here "-c" is used to create tar file "v" is used to be verbose and "f" is used to tell tar file name. You can see the tar file here

```
stock_trader@system:~/test ls -lrt
-r--r--r--  1 stock_trader Domain Users   0 Jul 15 11:42 equity
drwxrwxrwx+ 1 stock_trader Domain Users   0 Jul 15 14:33 stocks/
-r--r--r--  1 stock_trader Domain Users   0 Jul 15 15:30 currency
-rw-r--r--  1 stock_trader Domain Users 10K Jul 18 12:29 trading.tar
```

How to view contents of tar file in Unix or Linux

In earlier example of tar command in Unix or Linux we have created a uncompressed tar file called "trading.tar" now in this example we will see the actual content of that tar file.

```
stock_trader@system:~/test tar -tvf trading.tar
-r--r--r-- stock_trader/Domain Users 0 2011-07-15 15:30 currency
-r--r--r-- stock_trader/Domain Users 0 2011-07-15 11:42 equity
drwxrwxrwx stock_trader/Domain Users 0 2011-07-15 14:33 stocks/
-rwxrwxrwx stock_trader/Domain Users 0 2011-07-15 14:33 stocks/online_stock_exchanges.txt
```

here option "t" is used to display content of tar file in unix while options "v" and "f" are for "verbose" and "following". now you can clearly see that all the files which we wanted to be included in tar file are there.

How to extract contents from a tar file in Unix

In this example of unix tar command we will see how to extract files or directories from a tar file in unix or Linux. We will use same trading.tar file created in earlier example. In this example we will create a directory "trading" and

extract contents of trading.tar on that directory.

```
stock_trader@system:~/test/new ls -lrt
total 12K
-rw-r--r-- 1 stock_trader Domain Users 10K Jul 18 12:37 trading.tar
```

Now the directory is empty just trading.tar file

```
stock_trader@system:~/test/new tar -xvf trading.tar
currency
equity
stocks/
stocks/online_stock_exchanges.txt
```

This unix tar command will extract content of trading.tar in current directory. "x" is used for extracting. "v" is again for verbose and optional parameter in all our example.

```
stock_trader@system:~/test/new ls -lrt
-r--r--r--  1 stock_trader Domain Users   0 Jul 15 11:42 equity
drwxr-xr-x+ 1 stock_trader Domain Users   0 Jul 15 14:33 stocks/
-r--r--r--  1 stock_trader Domain Users   0 Jul 15 15:30 currency
-rw-r--r--  1 stock_trader Domain Users 10K Jul 18 12:37 trading.tar
```

Now you can see that all the files and directories which were included in tar file (stocks, equity and currency) has been extracted successfully.

How to create tar file in Unix with just specified contents

In above example of tar command in unix we have created tar file with all the contents available in current directory but we can also create tar file with selective content as shown in above example.

Now in our current directory we have both files and directories and we just want to include two files equity and currency in our tar file.

```
stock_trader@system:~/test ls -lrt
-r--r--r--  1 stock_trader Domain Users   0 Jul 15 11:42 equity
drwxrwxrwx+ 1 stock_trader Domain Users   0 Jul 15 14:33 stocks/
-r--r--r--  1 stock_trader Domain Users   0 Jul 15 15:30 currency
-rw-r--r--  1 stock_trader Domain Users 10K Jul 18 12:29 trading.tar
drwxr-xr-x+ 1 stock_trader Domain Users   0 Jul 18 12:46 new/

stock_trader@system:~/test tar -cvf equitytrading.tar equity currency
equity
currency
```

you see only two files equity and currency are included in our tar file.

How to create compressed tar file using gzip in Unix

In our previous example of Linux tar command we have created uncompressed tar file but most of the time we also need to create compressed tar file using gzip or bzip2. In this example of tar command in Linux we will learn about creating tar file using gzip.

```
stock_trader@system:~/test tar -zcvf trading.tgz *
currency
equity
stocks/
stocks/online_stock_exchanges.txt
```

you see creating tar file with gzip is very easy just use "-z" option and it will crate a gzip tar. .tgz or tar.gz extension is used to denote tar file with gzip. size of a compressed tar file is far less than uncompressed one.

```
stock_trader@system:~/test ls -lrt
-r--r--r--  1 stock_trader Domain Users   0 Jul 15 11:42 equity
drwxrwxrwx+ 1 stock_trader Domain Users   0 Jul 15 14:33 stocks/
-r--r--r--  1 stock_trader Domain Users   0 Jul 15 15:30 currency
-rw-r--r--  1 stock_trader Domain Users 219 Jul 18 13:01 trading.tgz
```

you can also view contents of gzip tar file by using earlier command in combination of "z" option and same is true for extracting content from gzip tar. below examples of unix tar command will show how to view contents of .tgz or .tar.gz file in unix.

```
stock_trader@system:~/test tar -ztvf trading.tgz
-r--r--r-- stock_trader/Domain Users 0 2011-07-15 15:30 currency
-r--r--r-- stock_trader/Domain Users 0 2011-07-15 11:42 equity
drwxrwxrwx stock_trader/Domain Users 0 2011-07-15 14:33 stocks/
-rwxrwxrwx stock_trader/Domain Users 0 2011-07-15 14:33 stocks/online_stock_exchanges.txt
```

Similarly we can extract contents from a **.tgz or .tar.gz file** as shown in below example of unix tar command :

```
stock_trader@system:~/test/new tar -zxvf trading.tgz
currency
equity
stocks/
stocks/online_stock_exchanges.txt

stock_trader@system:~/test/new ls -lrt
-r--r--r--  1 stock_trader Domain Users   0 Jul 15 11:42 equity
drwxr-xr-x+ 1 stock_trader Domain Users   0 Jul 15 14:33 stocks/
-r--r--r--  1 stock_trader Domain Users   0 Jul 15 15:30 currency
-rw-r--r--  1 stock_trader Domain Users 219 Jul 18 13:07 trading.tgz
```

How to create compressed tar file using bzip2 in Unix

----

bzip2 is another compression option we have which we can use with unix tar command. its exactly similar with our earlier option of compressing using gzip but instead of "z" option we need to use "j" tar option to create bzip2 file as shown in below example of tar command in unix.

```
stock_trader@system:~/test tar -jcvf trading.tar.bz2 *
currency
equity
stocks/
stocks/online_stock_exchanges.txt

stock_trader@system:~/test ls -lrt trading.tar.bz2
```

```
-rw-r--r--  1 stock_trader Domain Users 593 Jul 18 13:11 trading.tar.bz2
```

.tar.bz2 is used to denote a tar file with bzip2 compression. for viewing contents of bzip2 tar file and extracting content we can use as shown in ***example of UNIX tar command*** with gzip compression, just replace "-z" with "-j" for bzip2.

How to extract a particular file form .tar, .tar.gz or .tar.bzip2

In previous examples of extracting contetns from tar file we have extracted everything. sometime we just need a specific file from tar file. in this example of unix tar command we will extract a particular file from a tar archive.

```
stock_trader@system:~/test/new tar -jxvf trading.tar.bz2 equity
equity
```

its simple just specify name of file in this case its "equity". if your tar file is gzip one then use "-z" that's it. You can also use combination of grep and find command with tar to get more dynamic use.

How to extract group of file or directory from form .tar, .tar.gz or .tar.bzip2 in UNIX
---
you can extract a group of file form .tar, .tar.gz or .tar.bzip2 in Unix by specifying a matching pattern and using option "--wildcards". let's an example of tar command in unix with --wildcards

```
stock_trader@system:~/test/new tar -jxvf trading.tar.bz2 --wildcards "s*"
stocks/
stocks/online_stock_exchanges.txt
```

In above example of UNIX tar command we are extracting all files or directory which names starts with "s".

How to update existing tar file in Linux
--
You can also update or append new files in already created tar file. option"-r" is used for that. Let’s see an example of updatating tar file using tar command in UNIX:

```
stock_trader@system:~/test tar -cvf sample.tar equity currency
equity
currency

stock_trader@system:~/test tar -rvf sample.tar gold
gold

stock_trader@system:~/test tar -tvf sample.tar
-r--r--r-- stock_trader/Domain Users 0 2011-07-15 11:42 equity
-r--r--r-- stock_trader/Domain Users 221 2011-07-18 13:10 currency
-rw-r--r-- stock_trader/Domain Users   0 2011-07-18 13:30 gold
```

Apparently can not update compressed archives.if you try to do you will get error **"tar: Cannot update compressed archives"**

Calculating size of tar file in UNIX

Some time its useful to know the size of tar file before creating it and you can get it by using unix tar command as shown in below example:

```
stock_trader@system:~/test tar -cf - * | wc -c
20480
```

Size shown here is in KB and you can also calculate size for compressed tar file by using "z" for gzip and "j" for bzip2

That's all on this series of **10 example of tar command in UNIX or Linux**. If you guys have some other good example of UNIX tar command then please share with us via commenting.