**10 Useful Chaining Operators in Linux with Practical Examples**

Chaining of Linux commands means, combining several commands and make them execute based upon the behaviour of operator used in between them. Chaining of commands in Linux, is something like you are writing [short shell scripts](https://www.tecmint.com/category/bash-shell/) at the shell itself, and executing them from the terminal directly. Chaining makes it possible to automate the process. Moreover, an unattended machine can function in a much systematic way with the help of chaining operators.

Operators in Linux

**Read Also:** [How to Use Awk and Regular Expressions to Filter Text in Files](https://www.tecmint.com/use-linux-awk-command-to-filter-text-string-in-files/)

This Article aims at throwing light on frequently used **command­-chaining operators**, with short descriptions and corresponding examples which surely will increase your productivity and lets you write short and meaningful codes beside reducing system load, at times.

**1. Ampersand Operator (&)**

The function of ‘**&**‘ is to make the command run in background. Just type the command followed with a white space and ‘**&**‘. You can execute more than one command in the background, in a single go.

Run one command in the background:

tecmint@localhost:~$ ping ­c5 www.tecmint.com &

Run two command in background, simultaneously:

root@localhost:/home/tecmint# apt-get update & apt-get upgrade &

**2. semi-colon Operator (;)**

The semi-colon operator makes it possible to run, several commands in a single go and the execution of command occurs sequentially.

root@localhost:/home/tecmint# apt-get update ; apt-get upgrade ; mkdir test

The above command combination will first execute **update** instruction, then **upgrade** instruction and finally will create a ‘**test**‘ directory under the current working directory.

**3. AND Operator (&&)**

The **AND Operator** (**&&**) would execute the second command only, if the execution of first command **SUCCEEDS**, i.e., the exit status of the first command is **0**. This command is very useful in checking the execution status of last command.

For example, I want to visit website **tecmint.com** using [links command](https://www.tecmint.com/command-line-web-browsers/), in terminal but before that I need to check if the host is **live** or **not**.

root@localhost:/home/tecmint# ping -c3 www.tecmint.com && links www.tecmint.com

**4. OR Operator (||)**

The **OR Operator** (**||**) is much like an ‘**else**‘ statement in programming. The above operator allow you to execute second command only if the execution of first command fails, i.e., the exit status of first command is ‘**1**‘.

For example, I want to execute ‘**apt-get update**‘ from non-root account and if the first command fails, then the second **‘links www.tecmint.com**‘ command will execute.

tecmint@localhost:~$ apt-get update || links tecmint.com

In the above command, since the **user** was not allowed to **update** system, it means that the exit status of first command is **‘1’** and hence the last command ‘**links tecmint.com**‘ gets executed.

What if the first command is executed successfully, with an exit status ‘**0**‘? Obviously! Second command won’t execute.

tecmint@localhost:~$ mkdir test || links tecmint.com

Here, the user creates a folder ‘**test**‘ in his home directory, for which user is permitted. The command executed successfully giving an exit status ‘**0**‘ and hence the last part of the command is not executed.

**5. NOT Operator (!)**

The **NOT Operator** (**!**) is much like an ‘**except**‘ statement. This command will execute all except the condition provided. To understand this, create a directory ‘**tecmint**‘ in your home directory and ‘**cd**‘ to it.

tecmint@localhost:~$ mkdir tecmint

tecmint@localhost:~$ cd tecmint

Next, create several types of files in the folder ‘**tecmint**‘.

tecmint@localhost:~/tecmint$ touch a.doc b.doc a.pdf b.pdf a.xml b.xml a.html b.html

See we’ve created all the new files within the folder ‘**tecmint**‘.

tecmint@localhost:~/tecmint$ ls

a.doc a.html a.pdf a.xml b.doc b.html b.pdf b.xml

Now delete all the files except ‘**html**‘ file all at once, in a smart way.

tecmint@localhost:~/tecmint$ rm -r !(\*.html)

Just to verify, last execution. List all of the available files using [ls command](https://www.tecmint.com/15-basic-ls-command-examples-in-linux/).

tecmint@localhost:~/tecmint$ ls

a.html b.html

**6. AND – OR operator (&& – ||)**

The above operator is actually a combination of ‘**AND**‘ and ‘**OR**‘ Operator. It is much like an ‘**if-else**‘ statement.

For example, let’s do ping to **tecmint.com**, if success echo ‘**Verified**‘ else echo ‘**Host Down**‘.

tecmint@localhost:~/tecmint$ ping -c3 www.tecmint.com && echo "Verified" || echo "Host Down"

**Sample Output**

PING www.tecmint.com (212.71.234.61) 56(84) bytes of data.

64 bytes from www.tecmint.com (212.71.234.61): icmp\_req=1 ttl=55 time=216 ms

64 bytes from www.tecmint.com (212.71.234.61): icmp\_req=2 ttl=55 time=224 ms

64 bytes from www.tecmint.com (212.71.234.61): icmp\_req=3 ttl=55 time=226 ms

--- www.tecmint.com ping statistics ---

3 packets transmitted, 3 received, 0% packet loss, time 2001ms

rtt min/avg/max/mdev = 216.960/222.789/226.423/4.199 ms

**Verified**

Now, disconnect your internet connection, and try same command again.

tecmint@localhost:~/tecmint$ ping -c3 www.tecmint.com && echo "verified" || echo "Host Down"

**Sample Output**

ping: unknown host www.tecmint.com

**Host Down**

**7. PIPE Operator (|)**

This **PIPE** operator is very useful where the output of first command acts as an input to the second command. For example, pipeline the output of ‘**ls -l**‘ to ‘**less**‘ and see the output of the command.

tecmint@localhost:~$ ls -l | less

**8. Command Combination Operator {}**

Combine two or more commands, the second command depends upon the execution of the first command.

For example, check if a directory ‘**bin**‘ is available or not, and output corresponding output.

tecmint@localhost:~$ [ -d bin ] || { echo Directory does not exist, creating directory now.; mkdir bin; } && echo Directory exists.

**9. Precedence Operator ()**

The Operator makes it possible to execute command in precedence order.

Command\_x1 &&Command\_x2 || Command\_x3 && Command\_x4.

In the above pseudo command, what if the **Command\_x1** fails? Neither of the **Command\_x2**, **Command\_x3**, **Command\_x4** would executed, for this we use **Precedence Operator**, as:

(Command\_x1 &&Command\_x2) || (Command\_x3 && Command\_x4)

In the above pseudo command, if **Command\_x1** fails, **Command\_x2** also fails but Still **Command\_x3** and **Command\_x4** executes depends upon exit status of **Command\_x3**.

**10. Concatenation Operator (\)**

The **Concatenation Operator** (**\**) as the name specifies, is used to concatenate large commands over several lines in the shell. For example, The below command will open text file **test(1).txt**.

tecmint@localhost:~/Downloads$ nano test\(1\).txt

That’s all for now. I am coming up with another interesting article very soon. Till then Stay tuned, healthy and connected to **Tecmint**. Don’t forget to give your Valuable feedback in our comment section.