### **What is Linux shell ?**

### **The shell is a program that reads the user commands, evaluates it, and then prints the result. For evaluating the command, the shell may execute other commands, or pass them to the ‘kernel’.**

### **C Shell, Bourne Shell and Korn Shell,bash shell are the most famous shells which are available with most of the Unix variants.**

Or

The Linux command line is provided by a program called the shell.

### **Type of shell**

1. Bourne Shell (sh)
2. Bourne Again Shell (Bash)
3. C Shell (csh)
4. Korn Shell (ksh)
5. Z Shell (zsh)
6. Fish Shell (fish)

### **Choosing a Shell**

* **Bash** is the most widely used and recommended for general-purpose scripting and command-line use.
* **Zsh** is great for interactive use and highly customizable environments.
* **Ksh** is preferred for performance-intensive tasks and backward compatibility with older Unix systems.
* **Fish** is ideal for new users due to its ease of use and modern features.

### **What is my Shell type**

You can check using echo $o

cat /etc/shells

# List of acceptable shells in terminal

/bin/bash

/bin/csh

/bin/dash

/bin/ksh

/bin/sh

/bin/tcsh

/bin/zsh

### **What is Shell Scripting?**

**Shell scripting** is a way to automate tasks in a Unix-based operating system by writing scripts using shell commands. These scripts are written in shell programming languages such as Bash (Bourne Again SHell), and they allow users to automate repetitive tasks, configure systems, perform batch processing, and manage system operations efficiently.

Or

A bash script is a series of commands written in a file. These are read and executed by the bash program. The program executes line by line.

## **How Do You Identify a Bash Script?**

### **File extension of** .sh**.**

By naming conventions, bash scripts end with a .sh. However, bash scripts can run perfectly fine without the sh extension.

### **Scripts start with a bash bang.**

Scripts are also identified with a shebang. Shebang is a combination of bash # and bang ! followed the bash shell path. This is the first line of the script. Shebang tells the shell to execute it via bash shell. Shebang is simply an absolute path to the bash interpreter.

Below is an example of the shebang statement.

**#! /bin/bash**

### **What is a Shebang?**

A **shebang** (also called a hashbang) is a character sequence at the beginning of a script that specifies the interpreter that should be used to execute the script. It consists of the characters #! followed by the path to the interpreter.

### **Purpose of Shebang**

The shebang tells the operating system which interpreter to use to parse and execute the script. Without a shebang, the script may not run correctly, as the system may not know which interpreter to use.

### **Shebang Syntax**

The general syntax of a shebang is:

#!/path/to/interpreter

### **Examples of Shebangs**

1. **Bash**:

#!/bin/bash

**2. Sh** (Bourne Shell):

#!/bin/sh

3. **Python**:

#!/usr/bin/python3

4. **Perl**:

#!/usr/bin/perl

### **Note :- It is not mandatory to write a shebang line but follow the standard .**

Scripts have execution rights for the user executing them.

An execution right is represented by x. In the example below, my user has the rwx (read, write, execute) rights for the file test\_script.sh

## **How to Create Your First Bash Script**

Let's create a simple script in bash that outputs Hello World.

### **Create a file named hello\_world.sh**

touch hello\_world.sh

### **Comments in Shell Scripting**

**Comments** in shell scripting are lines or portions of lines that are ignored by the shell interpreter. They are used to explain and document the code, making it easier to understand and maintain.

#### **Single-Line Comments**

Single-line comments begin with a # character and continue to the end of the line. Everything after the # on that line is treated as a comment and is ignored by the shell.

#!/bin/bash

# This is a single-line comment

echo "Hello, World!" # This is another single-line comment

#### **Multi-Line Comments**

Shell scripts do not have a built-in syntax for multi-line comments like some other programming languages. However, you can achieve the effect of multi-line comments using a few techniques:

Using a Here Document (Heredoc) Trick:

#!/bin/bash

<<COMMENT

This is a multi-line comment

It spans multiple lines

Everything between the COMMENT tags is ignored

COMMENT

echo "Hello, World!"

In this example, << COMMENT tells the shell to ignore everything until it encounters the word COMMENT again.

**What are Variables in shell scripting ??**

**In shell scripting, a variable is a symbolic name for a storage location in memory. Variables are used to store data that can be referenced and manipulated throughout the script. Here’s a basic overview of how to use variables in shell scripting, along with some examples:**

**variable\_name=value**

**Note that there should be no spaces around the = sign.**

### **Accessing Variables**

**To access the value stored in a variable, you prefix the variable name with a $ sign.**

### **Example 1: Basic Variable Usage**

**#!/bin/bash**

**# Defining a variable**

**greeting="Hello, World!"**

**# Accessing and printing the variable**

**echo $greeting**

**#!/bin/bash**

**# Using the date command and storing the result in a variable**

**current\_date=$(date)**

**# Printing the variable**

**echo "Current date and time: $current\_date"**

**#!/bin/bash**

**# Using the ls command and storing the result in a variable**

**directory\_listing=$(ls)**

**# Printing the variable**

**echo "Files in the current directory:"**

**echo "$directory\_listing"**

**#!/bin/bash**

**# Using the df command and storing the result in a variable**

**disk\_usage=$(df -h /)**

**# Printing the variable**

**echo "Disk usage for root filesystem:"**

**echo "$disk\_usage"**

**In shell scripting, using $() for command substitution allows you to execute a command and capture its output in a variable. This method is preferred because it is more readable and less error-prone compared to the older backticks method (`command`). Here's a detailed explanation:**

**Command substitution is a mechanism by which the output of a command can be substituted in place of the command itself. The syntax for command substitution is $(command). When the shell encounters this syntax, it executes the command within the parentheses and replaces the entire expression with the output of that command.**

**Let's take a look at an example using command substitution:**

**#!/bin/bash**

**# Using the date command and storing the result in a variable**

**current\_date=$(date)**

**# Printing the variable**

**echo "Current date and time: $current\_date"**

**Here's the same example using backticks:**

**​​#!/bin/bash**

**# Using the date command and storing the result in a variable**

**current\_date=`date`**

**# Printing the variable**

**echo "Current date and time: $current\_date"**

**Interview question :- Compare the use of backticks ‘ ‘and $() for command substitution in shell scripting.**

**Important command to Master shell scripting**

1) echo

The echo command is a built-in feature in [Linux](https://phoenixnap.com/kb/what-is-linux) that prints out its arguments as standard output. It is used to display text strings or the command results.

echo “Hello, World!”

“Hello, World!

The ‘echo’ command is used to display the values of a variable. Onesuch a variable is ‘HOME’. To check the value of a variable precede the variable with a $ sign.

$ echo $HOME

/home/raghu

### **Changing the Output Format**

The -e option is used with escape characters, as it enables their use in the output. The escape characters are useful for formatting output and adding special characters or effects to text displayed by the echo command. Escape characters used with the -e option are presented in the table below.

| **Escape Character** | **Description** |
| --- | --- |
| \\ | Displays a backslash character. |
| \a | Plays a sound alert when displaying the output. |
| \b | Removes all the spaces between the text. |
| \c | Omits any output following the escape character. |
| \n | Adds a newline character to the output, which signifies the end of one line of text and the beginning of a new line. |
| \r | Performs a carriage return, which moves the cursor to the beginning of the current line without advancing to the next line. |
| \t | Creates horizontal tab spaces. |
| \v | Creates vertical tab spaces. |

echo -e 'Hello, World! \c This is PNAP!'

Note: If you are using the -e option, enter your string enclosed in single quotation marks. This ensures that escape characters are interpreted correctly.

Use \n any time you want to move the output to a new line:

echo -e 'Hello, \nWorld, \nthis \nis \nPNAP!'

Add horizontal tab spaces by using \t:

echo -e 'Hello, \tWorld!'

Use \v to create vertical tab spaces:

echo -e 'Hello, \vWorld, \vthis \vis \vPNAP!'

### **Writing to a File and the Terminal**

The echo command piped with the [tee command](https://phoenixnap.com/kb/linux-tee) is used to simultaneously display content in the terminal and store it in a file. For example, run:

echo "Hello, world!" | tee output.txt

### **Displaying a Variable Value**

The echo command is also used to display variable values as output. For instance, to display the current user name, use:

echo $USER

Reference :- <https://phoenixnap.com/kb/echo-command-linux>

2) date

The date command in Linux is a versatile utility used to display, set, or manipulate the system’s date and time information from the command line.

**Formatting Date and Time**

**date +"%Y-%m-%d %H:%M:%S”**

**date +"Year: %Y, Month: %m, Day: %d"**

**date "+DATE: %D%nTIME: %T"**

**date +"Week number: %V Year: %y"**

**These are the most common formatting characters for the date command:**

* **%D – Display date as mm/dd/yy**
* **%Y – Year (e.g., 2020)**
* **%m – Month (01-12)**
* **%B – Long month name (e.g., November)**
* **%b – Short month name (e.g., Nov)**
* **%d – Day of month (e.g., 01)**
* **%j – Day of year (001-366)**
* **%u – Day of week (1-7)**
* **%A – Full weekday name (e.g., Friday)**
* **%a – Short weekday name (e.g., Fri)**
* **%H – Hour (00-23)**
* **%I – Hour (01-12)**
* **%M – Minute (00-59)**
* **%S – Second (00-60)**

**Reference :-** [**https://phoenixnap.com/kb/linux-date-command**](https://phoenixnap.com/kb/linux-date-command)

**To create a file with the current timestamp in its name using the date command**

**touch file\_$(date +"%Y%m%d\_%H%M%S")**

3) head command

Displays the first few lines of a file. By default, the ‘head’ command displays the first 10 lines of a file. But with -n option, the number of lines to be viewed can be specified.

$ head /etc/passwd

root:x:0:0:root:/root:/bin/bash

daemon:x:1:1:daemon:/usr/sbin:/bin/sh

bin:x:2:2:bin:/bin:/bin/sh

sys:x:3:3:sys:/dev:/bin/sh

sync:x:4:65534:sync:/bin:/bin/sync

games:x:5:60:games:/usr/games:/bin/sh

man:x:6:12:man:/var/cache/man:/bin/sh

lp:x:7:7:lp:/var/spool/lpd:/bin/sh

mail:x:8:8:mail:/var/mail:/bin/sh

news:x:9:9:news:/var/spool/news:/bin/sh

4) tail command

Similar to ‘head’; the ‘tail’ command shows the last 10 lines by default, and -n option is available as well.

$ tail -n 4 /etc/passwd

raghu:x:1000:1000:Raghu Sharma,,,:/home/raghu:/bin/bash

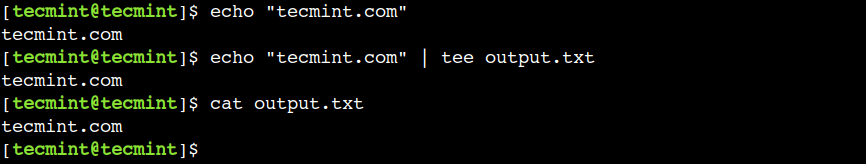
sshd:x:113:65534::/var/run/sshd:/usr/sbin/nologin

dictd:x:114:123:Dictd Server,,,:/var/lib/dictd:/bin/false

mysql:x:115:124:MySQL Server,,,:/nonexistent:/bin/false

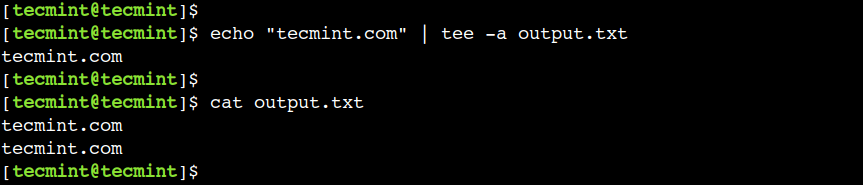
5) tee

tee command sends output to the standard output as well as the file.



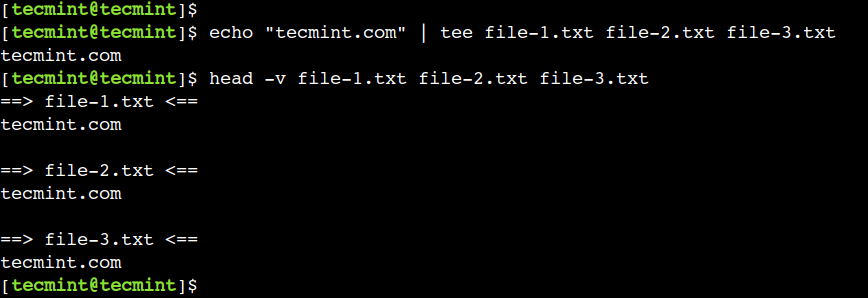
### **Append Output to File in Linux**

By default, the tee command overwrites the output files. However, we can avoid this by enabling the append mode, which appends the output at the end of the file:



### **Write Output to Multiple Files in Linux**

Similarly, we can use the tee command to write output to multiple files as shown.

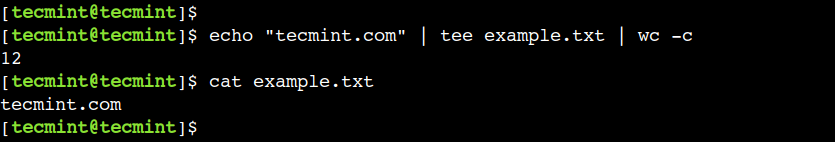


### **Send Output of One Command to Another**

Additionally, we can also use the tee command with the pipe operator (|). This method comes in handy when we want to store the output as well as forward it to another command.

$ echo "tecmint.com" | tee example.txt | wc -c

$ cat example.txt

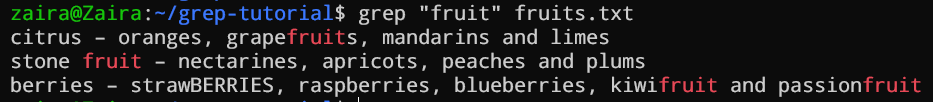


ls -l | tee filelist.txt

Reference :- https://www.tecmint.com/tee-command-examples/

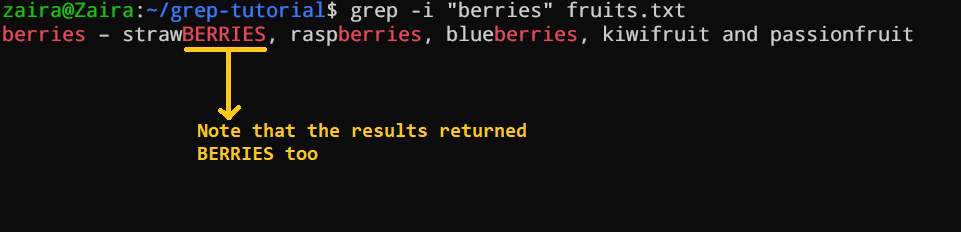
6) grep

Grep is a useful command to search for matching patterns in a file. grep is short for "global regular expression print".



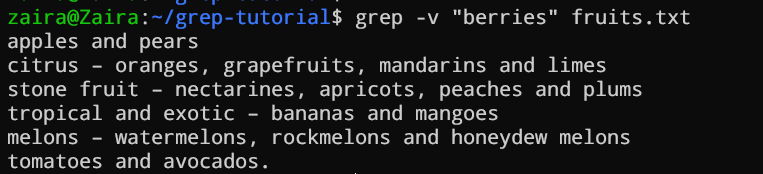
## **How to Ignore Case Distinctions using** -i

We can command grep to return results while ignoring the case of the matching string. Let's find the word "berries" from our sample file. It should match all occurrences of "berries" regardless of their case.



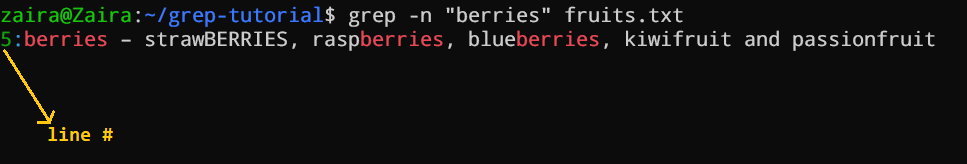
## **How to Select the Non-Matching Lines using** -v

We can reverse the results of the grep command to include non-matching results. Let's say, if we want to get all the lines that do not contain the word "berries", the command would look like this:



## **How to Find the Line Numbers Against Matching Input using** -n

**There are times when we want to get the line numbers against the matching string. For that, we can supply the -n flag to grep like this:**

****

7) sort

​​The sort command is a tool for sorting [file contents](https://phoenixnap.com/glossary/what-is-a-file) and printing the result in standard output. Reordering a file's contents numerically or alphabetically and arranging information in ascending or descending order improves readability.

The main rules are:

* Lines starting with a number go before the lines beginning with a letter, arranged in ascending order (1-10).
* Lines starting with letters are sorted in ascending alphabetical order (A-Z).

$ cat test.txt

Dr.B.R.Ambedkar

MahatmaJyotibaPhule

Budhha

ChatrapatiShahuMaharaj

budhha

Ramaai

$ sort test.txt

budhha

Budhha

ChatrapatiShahuMaharaj

Dr.B.R.Ambedkar

MahatmaJyotibaPhule

Ramaai

Sometimes, we need data in reverse order i.e., the opposite of alphabetical order. This is accomplished by using the -r option, as seen below:

$ sort test.txt

budhha

Budhha

ChatrapatiShahuMaharaj

Dr.B.R.Ambedkar

MahatmaJyotibaPhule

Ramaai

$ sort -r test.txt

Ramaai

MahatmaJyotibaPhule

Dr.B.R.Ambedkar

ChatrapatiShahuMaharaj

Budhha

budhha

Below, using the -n option, we've arranged the numbers in ascending order. Therefore the smallest number is at the top, and the largest number is at the bottom. We can also reverse the output using the same above option -r with -n and display the largest number at the top.

$ cat numeric.txt

14

04

34

1891

938

378

2356

$ sort -n numeric.txt

04

14

34

378

938

1891

2356

$ sort -nr numeric.txt

2356

1891

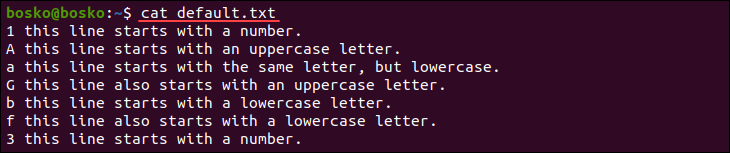
938

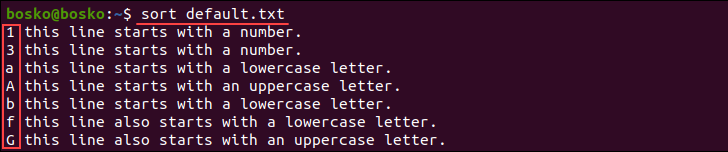
378

34

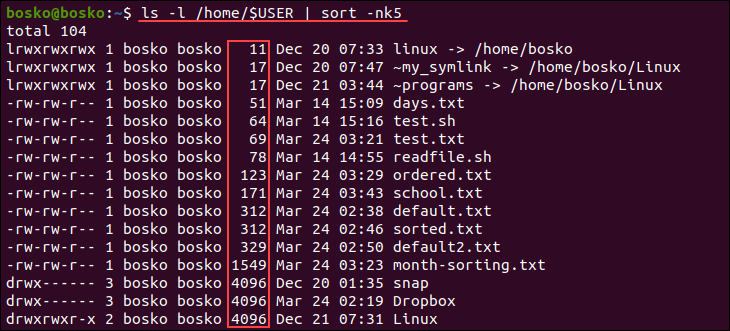
14

04





ls -l /home/$USER | sort -nk5



The -nk5 options specify numeric sorting in the fifth field.

Reference :- https://www.redhat.com/sysadmin/sort-command-linux

8) wc

The wc (word count) command in Linux is used to count the number of lines, words, and bytes (or characters) in a file or standard input.

### **Common Options**

* -l: Print the number of lines.
* -w: Print the number of words.
* -c: Print the number of bytes.
* -m: Print the number of characters.
* -L: Print the length of the longest line.

9) tr

[*tr*](http://man7.org/linux/man-pages/man1/tr.1.html) is short for “translate”.

**The *tr* command reads a byte stream from** [**standard input**](https://en.wikipedia.org/wiki/Standard_input) **(*stdin*), translates or deletes characters, then writes the result to the** [**standard output**](https://en.wikipedia.org/wiki/Standard_output) **(*stdout*).**

**tr [OPTION] SET1 [SET2]**

**We can use *tr* to perform text transformations like:**

* **character case conversion**
* **squeezing repeating characters**
* **deleting specific characters**
* **basic text replacement**

## **Convert Lowercase to Uppercase**

**cat baeldung.url**

**www.baeldung.com**

**tr 'a-z' 'A-Z' < baeldung.url**

**WWW.BAELDUNG.COM**

## **Basic Find and Replace**

**cat env.txt**

**$JAVA-HOME and $MAVEN-HOME are system variables.**

**cat env.txt | tr '-' '\_'**

**$JAVA\_HOME and $MAVEN\_HOME are system variables.**

**echo "{baeldung}" | tr '{}' '()'**

**(baeldung)**

**To delete specific characters use the -d option. This option deletes characters in the first set specified.**

**$ echo "Welcome To GeeksforGeeks" | tr -d W**

**Output**

**elcome To GeeksforGeeks**

**Reference :-** [**https://www.baeldung.com/linux/tr-command#findandreplace**](https://www.baeldung.com/linux/tr-command#findandreplace)

[**https://www.geeksforgeeks.org/tr-command-in-unix-linux-with-examples/**](https://www.geeksforgeeks.org/tr-command-in-unix-linux-with-examples/)

**10) find**

**11) locate**

**12) who**

**13) finger**

**14) hostname**