Advanced Linux Commands for DevOps Beginners

Lets discuss some more Linux commands which is kind of interesting and important

1) Find
2) Sed
3) Tr
4) Add user
5) File permissions
6) Awk

1. **find :** The find command is used to search for files and directories based on different conditions such as file type, name, size, and time of creation or modification. You can search files created or modified days ago or even minutes ago.

   Syntax: find <location> <options> <file_name_or_type>

   Examples:

   - find ~ -type f -name "code.txt"
     Searches for a file named code.txt in the home directory.
   - find / -type d -mtime 3 -name "india"
     Searches for a directory named india that was modified exactly 3 days ago.
   - find / -type d -mtime -3 -name "india"
     Searches for a directory named india modified within the last 3 days.
   - find / -type d -mtime +3 -name "india"
     Searches for a directory named india modified more than 3 days ago.
   - find / -type d -mmin 3 -iname "india"
     Searches for a directory named india modified exactly 3 minutes ago.
     Here, -iname means the search is case-insensitive.
   - find / -type d -mmin -5 -iname "india"
     Searches for a directory modified within the last 5 minutes.
   - find / -type d -mmin +5 -iname "india"
     Searches for a directory modified more than 5 minutes ago.

   Key Points:

   - -type f → files
   - -type d → directories
   - -mtime → modification time (in days)
   - -mmin → modification time (in minutes)
   - -name → case-sensitive search
   - -iname → case-insensitive search

2. **Sed: (Stream EDtionr):** The sed command (Stream Editor) is used to modify text in files without opening them in an editor. It works line by line, searches for patterns, and performs actions like replace, delete, insert, or print lines.
   By default, sed does not change the original file. It only prints the modified output to the terminal. To permanently modify the file, we must use the -i (in-place) option. Be careful when using -i because it directly edits the file.

   Format: sed [OPTIONS] 'command' filename

Examples

Substitute text

- sed 's/old/new/' file.txt
  s → substitute
  Replaces **only the first occurrence** of old in each line.
- **Global replacement**
  sed 's/dinesh/Dinesh/g' file.txt
  g → global
  Replaces **all occurrences** of dinesh in every line.
- **Edit the original file (in-place)**
  sed -i 's/dinesh/Dinesh/g' file.txt
  -i → edits the original file directly
  This is a **dangerous option** if used incorrectly.
- **Print specific lines using -n and p**
  sed -n '2p' file.txt
  -n → suppress automatic output
  p → print
  Prints **only the 2nd line**
  sed -n '2,5p' file.txt
  Prints lines **from 2 to 5**
- **Replace and print only**
  sed -n 's/dinesh/Dinesh/gp' file.txt
  Replaces text **globally**
  Prints only the modified lines
- **Delete lines**
  sed '2d' file.txt
  Deletes the **2nd line**
  sed '2,20d' file.txt
  Deletes lines **from 2 to 20**
- **Replace text in a specific line**
  sed '2s/dinesh/Dinesh/' file.txt

Replaces dinesh with Dinesh **only in line**

- **Important Concept: Word Boundary**

  Suppose the file contains:

  engineer
  engineering

  If you run:

  sed 's/engineer/engineering/' file.txt

  Output:

  engineering
  engineeringing

  This happens because sed finds engineer **inside** engineering.

  Correct way (match whole word only):

  sed 's/\<engineer\>/engineering/' file.txt

  \< and \> ensure **exact word matching**

  Only engineer is replaced, not engineering

  sed 's/\<engineer\>/engineering/' file.txt– so it will search for the entire word

Option	Explanation
s	Substitute the text
g	Global edit
p	Print
-i	Make the changes and edit the originalfile
d	Delete the line
-n	Not print the output
a	Append after the line
i	Insert before the line
c	Change the existing line

3. **tr**: The tr command is used to **translate or transform text** that comes as **output from another command**.

   It works on **standard input**, so it is usually used with a pipe (|).

   tr does not read files directly — it processes text passed to it.

   Syntax:

   command | tr 'set1' 'set2'

   Examples

   - **Convert lowercase to uppercase**

     cat demo.txt | tr 'a-z' 'A-Z'

     Converts all lowercase letters in demo.txt to uppercase.

   - **Convert uppercase to lowercase**

     cat demo.txt | tr 'A-Z' 'a-z'

Useful tr Options

Remove characters

cat demo.txt | tr -d '0-9'

Removes all numbers from the output.

Squeeze repeated characters

cat demo.txt | tr -s ' '

Replaces multiple spaces with a single space.

When to use tr

Change text case (upper ↔ lower)

Remove unwanted characters

Clean up output from other commands

4. **adduser**: The adduser command is a **simple and user-friendly way** to create a new user in Linux.

Example:

adduser dinesh

This command creates a user named **dinesh**.

During the process, it will:

Create the user

Create a home directory

Ask for and set the password

Ask for basic user details (optional)

adduser vs useradd

adduser

High-level and interactive

Easy to use

Prompts for password and user details

useradd

Low-level command

Requires more options

Does **not** set a password automatically

When you use useradd, you must manually set the password using:

passwd username

Example:

passwd dinesh

When to use adduser

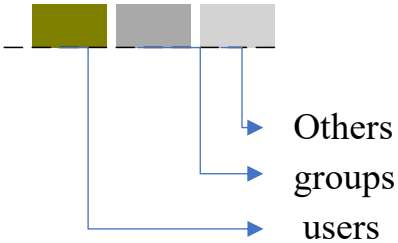
When you want a **quick and simple** way to create users

When you prefer an **interactive setup**

5. **chmod:** chmod is an important Linux command used to **change file or directory permissions**. It controls **who can read, write, or execute** a file or folder.
   Permissions are given to:
   User (owner)
   Group
   Others
   syntax: chmod <……..> file name

   d- directory
   r – read
   w - write
   x- execute
   l - link
   for file it will show like this -rwxrwxrwx

   d r w x r w x r w x demo ------- this demo folder has all the permeations



   chmod u+rwx fie.txt here you are giving all the permeations to the users
   chmod g+rwx you are giving all permeations to groups
   chmod o+ rwx you are giving all permeations to the others
   you can give what ever mermeats you want to
   give reading permeates for others : chmod o +r file.txt

there is another way to give permeations

   Numeric (octal) method
   Each permission has a value:
   Read (r) = 4
   Write (w) = 2
   Execute (x) = 1

Total:

   rwx = 4 + 2 + 1 = **7**

d r w x r w x r w x

7 7 7

Examples:

Give **all permissions to everyone**:

chmod 777 file.txt

Give **full permissions to user and group, execute only to others**:

chmod 771 file.txt

Give **read and write permissions only to user**:

chmod 600 file.tx

Quick tip

chmod is powerful. Giving 777 permissions can be **dangerous**, especially on servers. Use it **only when necessary**.

6. Awk: awk is a **text processing tool**.
   Think of awk as a **smart filter** That reads **line by line** Splits each line into **columns** Lets you **print, calculate, and format data** If sed edits text, awk **understands structure (columns)**.

   When do we use awk in real life?

   - Logs analysis
   - CSV files
   - Command output (ps, df, free)
   - Reports
   - Monitoring scripts

     Example:

     ps aux | awk '{print $1, $2}'

   Basic syntax

   awk 'pattern { action }' file

   Or:

   command | awk 'pattern { action }'

   If pattern is true → action runs.

   How awk works internally

   For every line:

1. Read the line
2. Split it into fields
3. Execute action
4. Move to next line

   Fields and records (MOST IMPORTANT)

   Record → one line

   Field → one column

Default separator = **space**

Symbol	Meaning
$0	whole line
$1	first column
$2	second column
$NF	last column
NF	number of fields
NR	line number

First basic example

File: data.txt

Dinesh Linux 50000
Ravi DevOps 60000
Kumar AWS 55000

Print whole line

```
awk '{print $0}' data.txt
```

Print specific columns

```
awk '{print $1, $3}' data.txt
```

Output:
Dinesh 50000
Ravi 60000
Kumar 55000

Print with text (formatting)

```
awk '{print "Name:", $1, "Salary:", $3}' data.txt
```

Conditions (very powerful)

Print only salary > 55000

```
awk '$3 > 55000 {print $1, $3}' data.txt
```

Print lines where department is DevOps

```
awk '$2 == "DevOps" {print $0}' data.txt
```

BEGIN and END blocks

BEGIN → runs once before reading file

END → runs once after file ends

```
awk 'BEGIN {print "Report Start"}
    {print $1, $3}
    END {print "Report End"}' data.txt
```

Output: Report Start
Dinesh 50000
Ravi 60000
Kumar 55000
Report End

You can use this in the csv files, which is Comma separated value

Field separator (-F)

CSV file

Dinesh,Linux,50000
Ravi,DevOps,60000

Command:

awk -F',' '{print $1, $3}' file.csv ---here F means file separator and after that you need to mention what is the file separator value. In this case it is ‘,’ Comma.

If you have values like **40%**, **20%**, and similar data, you can remove the **% symbol** using the awk command.

Example file: file.txt

40%
55%
80%

Command:

```
awk '{ gsub(/%/, "", $0); print }' file.txt
```

Explanation:

- gsub() stands for **global substitution**
- It searches for the % symbol and **removes it from each line**
- $0 represents the **entire line**
- print displays the modified output

Output:

40
55
80

Real-time use case:

This is commonly used when extracting **memory or disk usage** values, which are often shown as percentages (like 40%).

To process or calculate these values, the % symbol must be removed — and gsub makes this easy.

In short, **gsub helps clean unwanted characters from command output or files**, making the data usable for further processing.