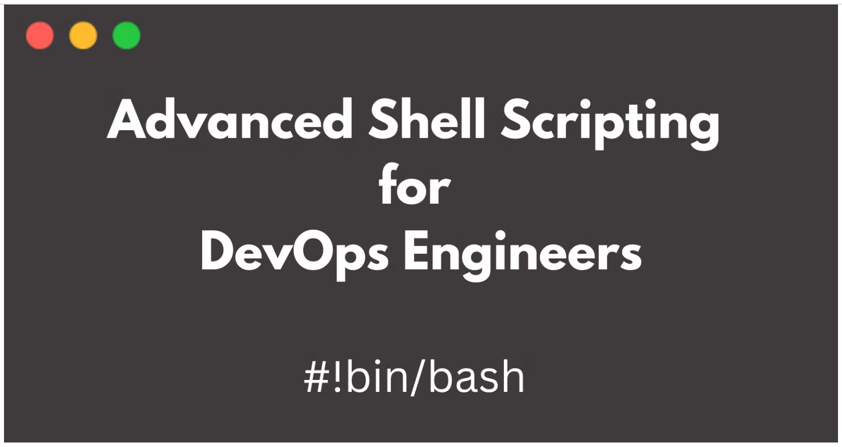
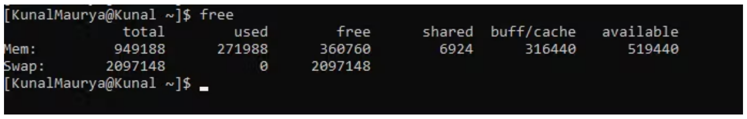


Day 5 Task: Advanced Linux Shell Scripting for DevOps Engineers

Kunal Maurya · Feb 27, 2023 · 7 min read

Memory Commands

1. `free`: The `free` command is a utility in Linux that shows the amount of free and used system memory, including the amount of memory used by buffers and caches. It also shows the swap space usage, if any.
   example:



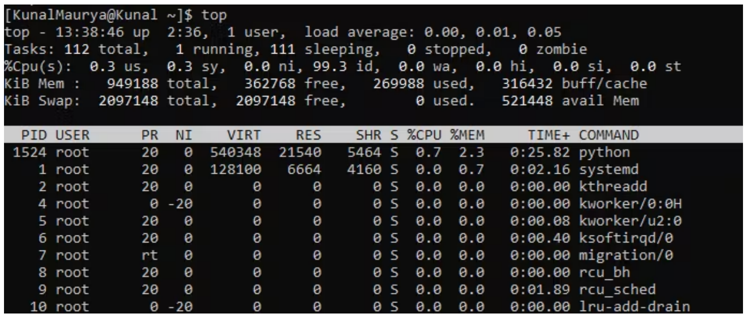
```
[KunalMaurya@Kunal ~]$ free
              total        used        free      shared  buff/cache   available
Mem:         949188      271988      360760        6924      316440      519440
Swap:       2097148           0     2097148
[KunalMaurya@Kunal ~]$ _
```

   This output shows the following information:

   - `total`: the total amount of physical memory in the system
   - `used`: the amount of physical memory currently in use
   - `free`: the amount of physical memory currently free
   - `shared`: the amount of shared memory in use
   - `buff/cache`: the amount of memory used by buffers and caches
   - `available`: the amount of memory that is available for allocation to new processes

The `Swap` section shows the swap space usage. In this example, there is 2097148 of swap space available, but none is currently being used.

1. `top`: `top` is a command-line utility that displays real-time information about system processes and resource usage. To display memory usage information with `top`, you can run:
   This command will display a list of running processes ordered by their memory usage, with the process using the most memory at the top.



```
[KunalMaurya@Kunal ~]$ top
top - 13:38:46 up  2:36,  1 user,  load average: 0.00, 0.01, 0.05
Tasks: 112 total,   1 running, 111 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.3 us,  0.3 sy,  0.0 ni, 99.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :   949188 total,   362768 free,   269988 used,   316432 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.   521448 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 1524 root      20   0  540348  21540   5464 S  0.7  2.3   0:25.82 python
    1 root      20   0  128100   6664   4160 S  0.0  0.7   0:02.16 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    4 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 kworker/0:0H
    5 root      20   0       0      0      0 S  0.0  0.0   0:00.08 kworker/u2:0
    6 root      20   0       0      0      0 S  0.0  0.0   0:00.40 ksoftirqd/0
    7 root      rt   0       0      0      0 S  0.0  0.0   0:00.00 migration/0
    8 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcu_bh
    9 root      20   0       0      0      0 S  0.0  0.0   0:01.89 rcu_sched
   10 root       0 -20       0      0      0 S  0.0  0.0   0:00.00 lru-add-drain
```

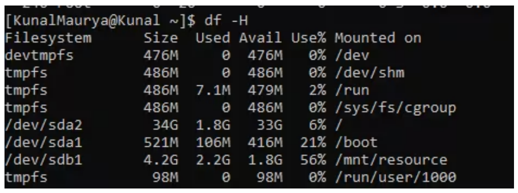
In this example output, you can see the memory usage information for each process, including the process ID (PID), user, %CPU usage, %MEM usage, and more.

2. `ps`: Displays information about running processes, including memory usage. example:

```
$ ps -eo pid,user,%mem,command --sort=-%mem | head
   PID USER     %MEM COMMAND
  1541 root      3.9 /usr/lib/xorg/Xorg -core :0 -seat seat0 -auth
  3762 john      1.5 /usr/bin/gnome-shell
  1682 root      0.8 /lib/systemd/systemd-journald
  1556 root      0.5 /usr/lib/upower/upowerd
  1651 root      0.4 /lib/systemd/systemd-udevd
  1648 root      0.4 /usr/sbin/rsyslogd -n
  1700 root      0.4 /usr/sbin/NetworkManager --no-daemon
```

3. `df`: Displays information about the file system disk space usage, including the total size, used space, and available space of mounted file systems.

```
$ df
Filesystem     1K-blocks     Used Available Use% Mounted on
/dev/sda1      10485760  4420664   6053084  43% /
tmpfs            817252       12    817240   1% /dev/shm
```



```
[KunalMaurya@Kunal ~]$ df -H
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        476M     0  476M   0% /dev
tmpfs           486M     0  486M   0% /dev/shm
tmpfs           486M  7.1M  479M   2% /run
tmpfs           486M     0  486M   0% /sys/fs/cgroup
/dev/sda2        34G  1.8G   33G   6% /
/dev/sda1       521M  106M  416M  21% /boot
/dev/sdb1       4.2G  2.2G  1.8G  56% /mnt/resource
tmpfs            98M     0   98M   0% /run/user/1000
```

4. `du`: Displays the disk usage of files and directories.

```
$ du -sh *
4.0K    dir1
4.0K    dir2
8.0K    dir3
24K     total
```

5. `fdisk`: A disk partitioning utility that allows you to create, delete, and modify disk partitions.

```
$ sudo fdisk -l
Disk /dev/sda: 20 GiB, 21474836480 bytes, 41943040 sectors
...

Device        Boot  Start      End  Sectors Size Id Type
/dev/sda1     *      2048 20971519 20969472  10G 83 Linux
/dev/sda2     20973566 41940991 20967426  10G  5 Extended
/dev/sda5     20973568 41940991 20967424  10G 8e Linux LVM
```

Advance Commands

1. `sed`: `sed` stands for stream editor and is used for text transformations on an input stream or file. It can perform a variety of operations such as replacing text, deleting lines, and inserting text.
   Example:

   ```
   $ cat file.txt
   Hello World!
   $ sed 's/World/Universe/' file.txt
   Hello Universe!
   ```

   In the above example, `sed` is used to replace the string "World" with "Universe" in the file `file.txt`.

2. `awk`: `awk` is a versatile text processing tool that is used to manipulate and transform data. It operates on a line-by-line basis and is typically used for data extraction, filtering, and reporting.
   Example:

   ```
   $ cat file.txt
   Name, Age
   John, 25
   Sarah, 30
   $ awk -F, '{print $1}' file.txt
   Name
   John
   Sarah
   ```

   In the above example, `awk` is used to print the first field (separated by comma) of each line in the file `file.txt`.

3. `grep`: `grep` is a command-line utility used to search for specific patterns in a file or input stream. It searches for a given pattern in one or more files and prints the lines that contain the pattern.
   Example:

   ```
   $ cat file.txt
   Hello World!
   Goodbye World!
   $ grep "Hello" file.txt
   Hello World!
   ```

In the above example, `grep` is used to search for the string "Hello" in the file `file.txt`.

4. `cut`: `cut` is used for extracting specific columns or fields from a text file or input stream. It takes a set of characters or fields from each line of a file or input stream and prints them to stdout.
   Example:

   ```
   $ cat file.txt
   Name, Age
   John, 25
   Sarah, 30
   $ cut -d, -f1 file.txt
   Name
   John
   Sarah
   ```

   In the above example, `cut` is used to extract the first field (separated by comma) of each line in the file `file.txt`.

5. `find`: `find` is a powerful utility used to search for files and directories in a directory hierarchy based on various criteria. It searches for files and directories recursively and performs actions on them.
   Example:

   ```
   $ find /var/log/ -name "*.log"
   /var/log/syslog
   /var/log/auth.log
   /var/log/syslog.1
   ```

   In the above example, `find` is used to search for all files ending with ".log" in the directory `/var/log/`.

6. `tar`: `tar` is used for creating and extracting archives in various formats. It can create and extract archives in formats such as tar, gzip, and bzip2.
   Example:

   ```
   $ tar -czvf archive.tar.gz folder/
   folder/
   folder/file1.txt
   folder/file2.txt
   $ tar -xzvf archive.tar.gz
   folder/
   folder/file1.txt
   folder/file2.txt
   ```

   In the above example, `tar` is used to create a compressed tar archive of the directory `folder/`. The second command extracts the archive to the current directory.

Backups – Shell Scripts

One of the simplest ways to backup a system is using a *shell script*. For example, a script can be used to configure which directories to backup, and pass those directories as arguments to the tar utility, which creates an archive file. The archive file can then be moved or copied to another location.

```
#!/bin/bash

# Set the source and destination directories
SOURCE_DIR="/home/user"
DESTINATION_DIR="/mnt/backup"

# Create the destination directory if it doesn't exist
mkdir -p $DESTINATION_DIR

# Set the filename and date for the backup
FILENAME="backup_$(date +%Y-%m-%d).tar.gz"

# Create the compressed archive of the source directory
tar -czvf "$DESTINATION_DIR/$FILENAME" $SOURCE_DIR

# Print a message when the backup is complete
echo "Backup complete: $FILENAME"
```

In this example, the script sets the source and destination directories as variables, creates the destination directory if it doesn't exist, generates a filename with the current date, creates a compressed archive of the source directory, and prints a message indicating that the backup is complete.

What is Cron?

Cron is a time-based job scheduler in Linux and Unix-like operating systems that allows you to schedule tasks to run automatically at specified intervals, such as once a day, once an hour, or even every minute.

Cron is a powerful tool for automating repetitive tasks, such as backups, system updates, and data processing.

What is Crontab?

Crontab, short for "cron table," is a configuration file used by the cron daemon to schedule tasks. The crontab file is typically stored in the /etc or /var/spool/cron directory, and each user has their own crontab file.

The crontab file contains a list of commands or scripts to be executed and the schedule for when each command should be run. Each line of the crontab file specifies a separate command or script, and the format of each line is as follows:

```
*     *     *     *     *  command to be executed
-     -     -     -     -
|     |     |     |     |
|     |     |     |     +----- day of the week (0 - 6) (Sunday = 0)
|     |     |     +------- month (1 - 12)
|     |     +--------- day of the month (1 - 31)
|     +----------- hour (0 - 23)
+------------- minute (0 - 59)
```

Open Crontab for root user using the following command:

```
$ crontab -e
```

Where -e tag is used to open the crontab in edit mode.

The Crontab file opens up.

Go at last of the Crontab and specify the command to be executed in following

Syntax:

```
(Minute) (Hour) (Date) (Month) (Day of Week) (Command)
```

- **Minute:** Minute at which the command is to be executed. It takes value from 0 to 59.
- **Hour:** Hour at which the command is to be executed. It takes value from 0 to 23.
- **Date:** Date of the month (1-31)
- **Month:** Month of the year (1-12)
- **Day of the Week:** It takes a value from 0 to 6.
  0: Sunday
  1: Monday etc.
- **Command:** Command to be executed

Schedule specified:

```
0 12 * * * ./backup.sh
0 12 * * * ./remove-backup.sh
```

The crontab executes the backup.sh file every day of the year at 12:00 pm and also finds and deletes the backup files that are older than five days.

User Management

User management in Linux involves creating, modifying, and deleting user accounts. This can be done using various command-line tools in Linux, such as useradd, usermod, and userdel.

Here are some examples of how to manage users in Linux:

1. Creating a new user account using the useradd command:

```
Copy codesudo useradd -m johndoe
```

This command creates a new user account named "Kunal" with a home directory (-m option) in the Linux system.

1. Modifying a user account using the usermod command:

```
Copy codesudo usermod -aG sudo Kunal
```

This command adds the user "Kunal" to the sudo group (-aG option) in the Linux system, which gives the user administrative privileges.

1. Deleting a user account using the userdel command:

```
Copy codesudo userdel -r Kunal
```

This command removes the user account "Kunal" and its home directory (-r option) from the Linux system.

Thank you for reading! Hope you find this article helpful.

~Kunal