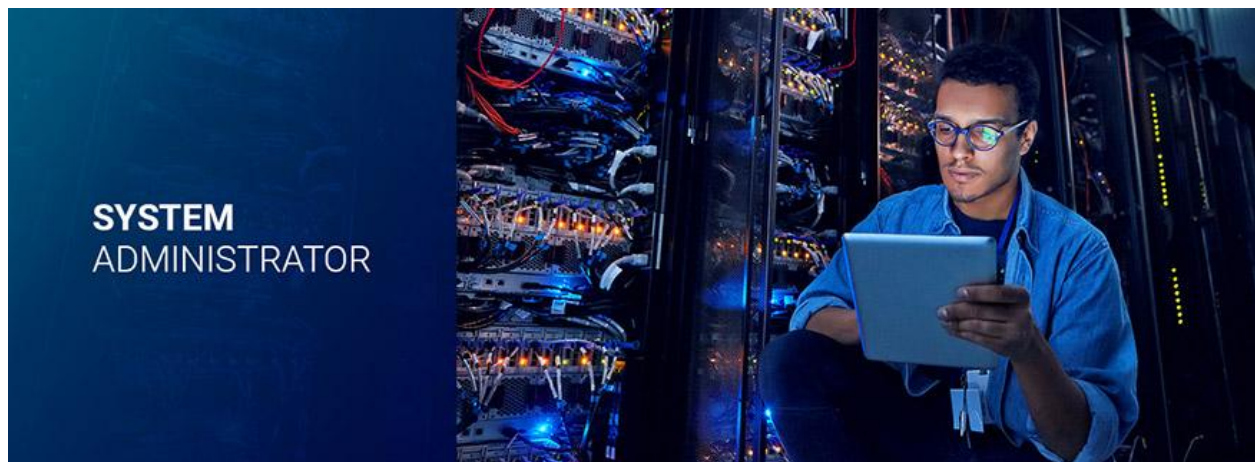


My Name is Amar Kumar, My Job is System Administration



BY

Amar Kumar (CED17I029)

Contents

- I. Acknowledgement
- II. Preface
- III. Chapters
- IV. References



I. Acknowledgement

First of all, I would like to thank our professor **Dr. Sankaran Vaidyanathan** for giving us the opportunity and inspiration to write this book.

I would like to thank my friends for the motivation and support they provided. Then I would like to thank my family members, without whom it was impossible for me to complete the book as they provided the source where I could write the book.

Then I would like to thank the internet as well for providing me the source and information needed for writing this book.

Thank You

Amar Kumar

IIITDM kancheepuram

Tamil Nadu, India - 600127

II. Preface

This book emerged from my exploration of a job as a System Administration. This book will cover all the important aspects of a System Administrator i.e. their roles and responsibilities. The reader of this book should have familiarity with the linux environment, Terminal etc.



III. Chapters

Introduction to System Administration

System

A System is a group of interacting or interrelated elements that act according to a set of rules to form a unified whole.

Administrator

A person whose job is to organize or manage a system, a business, etc.

System Administrator

A system administrator, or simply sysadmin in short, is a person who is responsible for the upkeep, configuration, and reliable operation of computer systems, especially multi-user computers, such as servers.

The system administrator seeks to ensure that the uptime, performance, resources and security of the computers they manage meet the needs of the users, without exceeding a set budget when doing so.

To meet these needs, a system administrator may acquire, install, or upgrade computer components and software, provide routine automation, maintain security policies, troubleshoot, train or supervise staff, or offer technical support for projects.

Some related fields


A system administrator is broadly categorized into database administrator, network administrator, security administrator, web administrator, computer operator, SRE(site reliability engineer).

Education requirements

Most employers require a bachelor's degree in a related field, such as computer science, information technology, electronics engineering, or computer engineering. Some schools also offer undergraduate degrees and graduate programs in system administration. An employee will be required to have some experience with the computer system they are expected to manage. In some cases, they may require some kind of industry certification.

Roles and responsibilities

- Analyzing system logs and identifying potential issues with computer systems.
- Applying operating system updates, patches, and configuration changes.
- Installing and configuring new hardware and software.
- Adding, removing, or updating user account information, resetting passwords, etc.
- Answering technical queries and assisting users.
- Responsibility for security.
- Responsibility for documenting the configuration of the system.
- Troubleshooting any reported problems.
- System performance tuning.
- Ensuring that the network infrastructure is up and running.
- Configuring, adding, and deleting file systems.
- Ensuring parity between dev, test and production environments.
- Training users
- Plan and manage the machine room environment



In larger organizations, some of the tasks above may be divided among different system administrators or members of different organizational groups. In smaller organizations, the system administrator might also act as technical support, Database Administrator, Network Administrator, Storage (SAN) Administrator or application analyst.

Skills required

- Knowledge of operating systems and applications
- Hardware and software troubleshooting
- Problem solving skills is perhaps the most important
- When a computer system goes down or malfunctions, they should be able to quickly and correctly diagnose what is wrong and how to fix it
- Knowledge of several programming languages used for scripting or automation of routine tasks
- A sysadmin must have a strong grasp of computer security

Introduction to Scripting


Most of the system administrators know scripting language. A scripting language is a programming language for a runtime system that automates the execution of tasks that would otherwise be performed individually by a human operator.

These scripts also help a user to understand the order of execution of a file and thus give more clarity about the code.

If you are familiar with language c/c++, you must be knowing about the main function in which we write the order of execution of different functions which helps to debug the code easily.

Below is an example of a scripting language. This is a .sh file which is unix (linux) shell executables files which had been written by me during my 7th semester for the course High performance computing.

```
D: > Amar > #College > sem7 > high-performance-computing-noor > lab3 > question2b > exp3_q2b_script.sh
1  g++ exp3_generate_random_number_q2b_ced17i029.cpp
2  ./a.out>exp3_output_random_number_q2b_ced17i029.txt
3  g++ exp3_q2b_ced17i029.cpp -fopenmp -o q2b
4  export OMP_NUM_THREADS=1;
5  ./q2b<exp3_output_random_number_q2b_ced17i029.txt
6  export OMP_NUM_THREADS=2;
7  ./q2b<exp3_output_random_number_q2b_ced17i029.txt
8  export OMP_NUM_THREADS=4;
9  ./q2b<exp3_output_random_number_q2b_ced17i029.txt
10 export OMP_NUM_THREADS=6;
11 ./q2b<exp3_output_random_number_q2b_ced17i029.txt
12 export OMP_NUM_THREADS=8;
13 ./q2b<exp3_output_random_number_q2b_ced17i029.txt
14 export OMP_NUM_THREADS=10;
15 ./q2b<exp3_output_random_number_q2b_ced17i029.txt
16 export OMP_NUM_THREADS=12;
17 ./q2b<exp3_output_random_number_q2b_ced17i029.txt
18 export OMP_NUM_THREADS=14;
19 ./q2b<exp3_output_random_number_q2b_ced17i029.txt
20 export OMP_NUM_THREADS=16;
21 ./q2b<exp3_output_random_number_q2b_ced17i029.txt
22 export OMP_NUM_THREADS=20;
23 ./q2b<exp3_output_random_number_q2b_ced17i029.txt
24 export OMP_NUM_THREADS=24;
25 ./q2b<exp3_output_random_number_q2b_ced17i029.txt
```



First of all this script is compiling a c++ file (line 1) and then executing it and storing the output in some .txt file (line 2). Then it is again compiling a c++ file and creating an object for this c++ file (line 3) and then setting the number of threads (line 4) and then executing the file for the same number of threads (line 5). Then line 4 and line 5 are repeated for different numbers of threads.

We can see here that if we need to write this command in a terminal then we need to spend a lot of time as we need to write one line and then execute it and again write the other line and execute it. But creating a shell script file is giving an advantage of automation of the task of executing the program.

Process of writing and executing a script

- Open terminal.
- Navigate to the place where you want to create script using '**cd**' command.
- **Cd** (enter) [This will bring the prompt at **Your home Directory**].
- touch **hello.sh** (Here we named the script as **hello**, remember the '**.sh**' extension is compulsory).
- vi **hello.sh** (nano **hello.sh** or code **hello.sh**) [You can use your favourite editor, to edit the script].
- **chmod +x hello.sh** (making the script executable).
- **sh hello.sh** or **./hello.sh** (running the script)

Shell Basics

Shell can also be used as a command editor, like using vim or emacs.

To get a Linux shell, you need to start a terminal (ctrl + alt + t).

To see what shell you have, run: echo **\$SHELL**. I have bash shell

For My Terminal it gives output as following:-

```
amar@amarkr:~$ echo $SHELL
/bin/bash
amar@amarkr:~$
```

In Linux, the dollar sign (\$) stands for a shell variable. The 'echo' command just returns whatever you type in. The pipeline instruction (|) comes to rescue, when chaining several commands.

Some of the useful commands in linux

cat - It helps us to create, view, concatenate files.

```
amar@amarkr:~/Desktop$ cat tran.txt
तब मैं ने उन से कहा , तुम्हारे परभु से क्षमा करो ; क्योंकि वह तुम्हें सचमुच क्षमा करता है।
वह तेरे लिये अकाश से बहुत सी वर्षा बरसाएगा , और तेरे साथ बहुत यत्न से अकाश से भेजेगा ।
और तुम्हारे लिये धन और पुत्र बढ़ाएंगे , और तुम्हें बारि यां और नाले देंगे।
जो तुम पर हुआ है , वह यह है कि परमेश्वर के महा तमय का भय नहीं मानना चाहिए।
```

wc - Count Number of Lines, Words, and Characters

```
amar@amarkr:~/Desktop$ wc mergeSort.cpp
 54  106 1051 mergeSort.cpp
```

We can use suffix with wc command like -l for getting only line count, -w for getting only word count, and -c for getting only character count.

```
amar@amarkr:~/Desktop$ wc -l mergeSort.cpp
54 mergeSort.cpp
amar@amarkr:~/Desktop$ wc -c mergeSort.cpp
1051 mergeSort.cpp
amar@amarkr:~/Desktop$ wc -w mergeSort.cpp
106 mergeSort.cpp
```

There are plenty of commands which are used frequently (like ls, cd, mkdir, touch etc.) and are not possible to list out all those.

Regular expressions

Ever wondered, how can we find all file names starting with **am**, that's where Regular expressions come into help, it's supported by most of the programming languages.

They're also used by UNIX commands such as **grep** and **vi**. They are so common that the name is usually shortened to "regex."

In the below command, i am searching all lines starting with **for** and here is the output

```
amar@amarkr:~/Desktop$ grep for *
Binary file a.out matches
bubble_sort.cpp:     for(int i=0 ; i<10 ; ++i){
bubble_sort.cpp:     for(int i=0 ; i<10 ; ++i){
bubble_sort.cpp:         for(int j=0 ; j<10-i-1 ; ++j){
```

Regular expressions are powerful, but they cannot recognize all possible grammars. Their most notable weakness is that they cannot recognize nested delimiters. For example, it's not possible to write a regular expression that recognizes valid arithmetic expressions when parentheses are allowed for grouping.

The matching process

Code that evaluates a regular expression attempts to match a single given text string to a single given pattern. The "text string" to match can be very long and can contain embedded newlines. It's often convenient to use a regex to match the contents of an entire file or HTML document

Matching is case sensitive.

Regular Expressions discussion is incomplete without perl programming language, perl has the most powerful library for regular expressions.

PERL PROGRAMMING

Perl, created by Larry Wall, was the first of the truly great scripting languages. It offers vastly more power than bash, and well-written Perl code is quite easy to read. On the other hand, Perl does not impose much stylistic discipline on developers, so Perl code written without regard for readability can be cryptic. Perl has been accused of being a write-only language.

Booting and shutting down

A system administrator knows all about the Linux from starting/booting the laptop till the laptop is shutdown. They know about the process involved between these. They know what happens just after pressing the power button. We will see all these in this chapter


Power-on to Working System

The sequence of steps to move a computer from the state of turned-off to a working operating system is commonly called **boot-strapping**. We start with nothing and incrementally give the computer more functionality.

We will not concern ourselves here with the matters of POST, BIOS and set up data. The BIOS loads a Boot Loader program from the master boot record (MBR) of the active partition. The Boot Loader program loads and starts the operating system, so this is where we'll begin.

Boot Loader

Before discussing boot loaders, we should note that many modern Linux distributions place the Linux kernel and other needed files on a small partition that usually becomes the **/boot/** directory on the booted system. This partition uses a basic file system, such as ext4, that the loader can read. Then once the kernel is loaded, it will be capable of working with more advanced file systems, such as LVM, for the root file system partition.



Older Linux systems used LILO as the boot loader. Newer systems use GRUB. Both will allow the user to pick between booting different Linux kernels or root file systems and can also boot different operating systems, such as Windows. The user may also use the boot loader to pass arguments to the kernel such as to boot into single user mode.

GRUB

Most Linux distributions installers will install GRUB, so it is usually not necessary to install it.

The configuration file for GRUB is usually ***/boot/grub/menu.lst***.


GRUB 2

In some Linux Systems we have a newer version of GRUB. GRUB2 which is intended to require less effort by the user to maintain. Its main configuration file, */boot/grub/grub.cfg* is not meant to be directly edited, but rather automatically generated by a set of shell scripts.

The shell scripts used by GRUB 2 are contained in the ***/etc/grub.d/*** directory. The main file of interest to users in this directory is ***40_custom***, which may be used to store custom menu entries and directives.

Service Starting and System Monitoring

After the kernel is loaded, the system can start the process of starting needed services and daemons so that we have a usable system and this is



done by **init** program. The init program started a sequence of shell scripts that configured the system and started daemon programs. After the system was completely booted, the role of init was mostly restricted to process monitoring.

In response to new hardware and the capabilities of more consumer oriented operating systems, such as Windows and Macintosh, many in the linux community felt that init was not proactive enough. In particular, they felt that it should be able to detect changes to the hardware environment, such as devices (jump drives) being plugged into the USB bus or network cables removed or plugged into the USB bus or network cables removed or plugged in. Currently to deal with this problem we are using a program called **upstart** and **systemd** as replacement to init.

The upstart program is mostly compatible with the traditional init configuration files, but systemd breaks backwards compatibility with init.

Run Levels

Each service or daemon has a shell script in the **/etc/rc.d/init.d/** directory that can be invoked to start, stop, restart, reload, or report the status of the service.

Systemd

Systemd is a system and service manager for Linux, compatible with SysV and LSB init scripts. Systemd provides:

- Aggressive parallelization capabilities
- Uses socket and D-Bus activation for starting services.
- Offers on-demand starting of daemons, keeps track of processes using Linux cgroups.
- Supports snapshotting and restoring of the system state.
- Maintains mount and auto mount points.
- Implements an elaborate transactional dependency-based service control logic.

The ***systemctl*** command is the primary tool to manage systemd. It combines the functionality of sysvinit's ***service*** and ***chkconfig*** commands into a single tool you can use to enable to disable services permanently or only for the current session.


Systemd manages so-called units, which are representations of system resources and services. This following list shows the unit types that systemd can manage.

For more read please visit:

<https://docs.fedoraproject.org/en-US/quick-docs/understanding-and-administering-systemd/index.html>

shutdown

The ***shutdown*** command brings the system down in a secure way. All ***logged-in*** users are notified that the system is going down, and login



operations are blocked. It is possible to shut the system down immediately, or after a delay.

For more read please visit:

<https://www.computerhope.com/unix/ushutdow.htm>

About Users in linux

Listing all users

Details of local users can be found in the **/etc/passwd** file. Every line contained in the file contains the information of one user.

```
amar@amarkr:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
```

Display the list of logged-in users


To display a list of the logged-in users and the information such as boot time, processes, hostnames, and more, use the **who** command.

```
amar@amarkr:~$ who
amar      :1                2021-04-14 00:41 (:1)
amar@amarkr:~$
```

Adding new users

To add new users we use **useradd** command. The general syntax for the useradd command is as follows:

```
useradd [OPTIONS] USERNAME
```



Only root users with sudo privileges can use the **useradd** command to create new user accounts.

When invoked, **useradd** creates a new user account according to the options specified on the command line and the default values set in the **/etc/default/useradd** file.

useradd also reads the content of the **/etc/login.defs** file. This file contains configuration for the shadow password suite such as password expiration policy, ranges of user IDs used when creating system and regular users, and more.

Different variation of user creation in Linux

- Create simple new user

To create a new user account, invoke the **useradd** command followed by the name of the user.

For example to create a new user named username you would run:

```
sudo useradd username
```

To be able to log in as the newly created user, you need to set the user password. To do that run the **passwd** command followed by the surname:

```
sudo passwd username
```

You will be prompted to enter and confirm the password. Make sure to use

a strong password.

Changing password **for** user username.

New password:

Retype **new** password:

passwd: all authentication tokens updated successfully.

- The other variants are
 - Add new user and create home directory.
 - Creating a user with a specific home directory.
 - Creating a user with specific user id.
 - Creating users with specific group id.
 - Creating users and assigning multiple groups.
 - Creating a user with a specific login shell.
 - Creating a user with Custom Comment.
 - Creating users with an Expiry Date.
 - Creating a system user.
 - Changing the default useradd values.

To read more about the procedure to create the above types of user please visit:

<https://linuxize.com/post/how-to-create-users-in-linux-using-the-useradd-command/>

Introduction to access control

Access control is an area of active research, and it has long been one of the major challenges of operating system design. Generally speaking, operating systems define accounts for individual users, and they offer those users a smorgasbord of possible operations: editing text files, logging into remote computers, setting the system's hostname, installing new software, and so on. The access control system is the black box that considers potential actions (user/operation pairs) and issues rulings as to whether each action is permissible.

Filesystem access control

In the traditional model, every file has both an owner and a group, sometimes referred to as the “**group owner**.” The owner can set the permissions of the file. In particular, the owner can set them so restrictively that no one else can access it.

Although the owner of a file is always a single person, many people can be group owners of the file, as long as they are all part of a single group. Groups are traditionally defined in the `/etc/group` file, but these days group information is more commonly stored on an **NIS** or **LDAP** server on the network.

Process ownership

The owner of a process can send the process signals and can also reduce (degrade) the process's scheduling priority. Processes actually have multiple identities associated with them: a real, effective, and saved UID; a real, effective, and saved GID; and under Linux, a **“filesystem UID”** that is used only to determine file access permissions.

The root account


The root account is UNIX's omnipotent administrative user. It's also known as the superuser account, although the actual username is **“root”**.

The defining characteristic of the root account is its **UID** of **0**. Nothing prevents you from changing the username on this account or from creating additional accounts whose UIDs are 0; however, these are both bad ideas. Such changes have a tendency to create inadvertent breaches of system security. They also create confusion when other people have to deal with the strange way you've configured your system.

Traditional UNIX allows the **superuser** (that is, any process whose effective UID is 0) to perform any valid operation on any file or process.

Setuid and setgid execution

Traditional UNIX access control is complemented by an identity substitution system that's implemented by the kernel and the filesystem in collaboration.




It allows specially prepared executable files to run with elevated permissions, usually those of **root**. This mechanism lets developers and administrators set up structured ways for unprivileged users to perform privileged operations

Role-based access control

Role-based access control, sometimes known as RBAC, is a theoretical model formalized in 1992 by David Ferraiolo and Rick Kuhn. The basic idea is to add a layer of indirection to access control calculations. Instead of permissions being assigned directly to users, they are assigned to intermediate constructs known as “roles,” and roles in turn are assigned to users. To make an access control decision, the access control library enumerates the roles of the current user and checks to see if any of those roles have the appropriate permissions

Logging in to the root account

Since root is just another user, you can log in directly to the root account and work your will upon the system. However, this turns out to be a bad idea. To begin with, it leaves no record of what operations were performed as root. That’s bad enough when you realize that you broke something last night at 3:00 a.m. and can’t remember what you changed; it’s even worse when an access was unauthorized and you are trying to figure out what an intruder has done to your system. Another disadvantage is that the log-in-as-root



scenario leaves no record of who was really doing the work. If several people have access to the root account, you won't be able to tell who used it and when. For these reasons, most systems allow root logins to be disabled on terminals, through window systems, and across the network—everywhere but on the system console.


su: substitute user identity

A marginally better way to access the root account is to use the **su** command. If invoked without arguments, **su** prompts for the root password and then starts up a root shell. Root privileges remain in effect until you terminate the shell by typing or the **exit** command. **su** doesn't record the commands executed as root, but it does create a log entry that states who became root and when.

Password vaults and password escrow

A password vault is a piece of software (or a combination of software and hardware) that stores passwords for your organization in a more secure fashion than “Would you like Windows to remember this password for you?” Several developments have made a password vault almost a necessity:

- The proliferation of passwords needed not just to log in to computers, but also to access web pages, configure routers and firewalls, and administer remote services

- 
- The increasing need for strong (read “not very memorable”) passwords as computers get so fast that weak passwords are easily broken
 - Regulations that require access to certain data to be traceable to a single person—no shared logins such as root

Process management

Process

A running program is called a Process. In simple terms, any command that you give to your linux machine starts a new process. **Example** - when you double click on the chrome icon to open it, we create a new process.

Types of process

1. Foreground Processes: They run on the screen and need input from the user. For example office programs
2. Background Processes: They run in the background and usually do not need user input. For example Antivirus or Daemon Processes.

Different Commands for Process Management in Linux

top - This utility tells the user about all the running processes on the linux machine.

```
top - 03:46:04 up 3:06, 1 user, load average: 0.51, 0.63, 0.64
Tasks: 371 total, 1 running, 370 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.3 us, 0.5 sy, 0.0 ni, 97.8 id, 0.5 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 7839.7 total, 2853.2 free, 3173.7 used, 1812.8 buff/cache
MiB Swap: 2048.0 total, 2048.0 free, 0.0 used, 4211.9 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1727	amar	20	0	4676868	443380	118456	S	8.0	5.5	7:42.08	gnome-+
1551	root	20	0	349616	115860	73300	S	3.7	1.4	3:06.76	Xorg
1127	root	-51	0	0	0	0	S	1.7	0.0	2:03.53	irq/14+
234	root	-51	0	0	0	0	S	1.3	0.0	0:15.00	irq/83+
238	root	-51	0	0	0	0	S	1.0	0.0	0:10.72	irq/12+
3094	amar	20	0	370416	105336	65792	S	0.7	1.3	1:08.36	chrome
3053	amar	20	0	1200156	340240	152360	S	0.3	4.2	5:47.39	chrome
3825	amar	20	0	36.5g	293464	102268	S	0.3	3.7	8:05.15	chrome
9405	root	20	0	0	0	0	I	0.3	0.0	0:00.03	kworke+
9743	amar	20	0	12124	4000	3240	R	0.3	0.0	0:00.08	top
9751	amar	20	0	2958568	45840	32640	S	0.3	0.6	0:00.30	org.gn+
1	root	20	0	167804	11628	8416	S	0.0	0.1	0:02.30	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kthrea+
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_pa+
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworke+

ps - This command stands for “**Process Status**”. It is similar to the “Task Manager” that pop-ups in a Windows Machine when we use **Ctrl+Alt+Del**. This command is similar to the “**top**” command but the information displayed is different.

```
amar@amarkr:~/Desktop$ ps ux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
amar	1462	0.0	0.1	19372	10720	?	Ss	00:41	0:00	/lib/systemd
amar	1463	0.0	0.0	169080	3564	?	S	00:41	0:00	(sd-pam)
amar	1468	0.0	0.2	2269260	20128	?	S<sl	00:41	0:00	/usr/bin/pul
amar	1470	0.0	0.3	511624	24792	?	SNsl	00:41	0:00	/usr/libexec
amar	1474	0.0	0.0	8584	5672	?	Ss	00:41	0:01	/usr/bin/dbu
amar	1489	0.0	0.0	239812	7912	?	Ssl	00:41	0:00	/usr/libexec
amar	1494	0.0	0.0	378344	6080	?	Sl	00:41	0:00	/usr/libexec
amar	1501	0.0	0.1	313936	9096	?	Ssl	00:41	0:00	/usr/libexec
amar	1507	0.0	0.0	238104	7240	?	Ssl	00:41	0:00	/usr/libexec
amar	1511	0.0	0.0	235700	6156	?	Ssl	00:41	0:00	/usr/libexec
amar	1515	0.0	0.1	316728	8732	?	Ssl	00:41	0:00	/usr/libexec
amar	1521	0.0	0.0	240176	7640	?	Sl	00:41	0:00	/usr/bin/gno
amar	1524	0.0	0.0	235876	6060	?	Ssl	00:41	0:00	/usr/libexec
amar	1528	0.0	0.4	546132	36232	?	Sl	00:41	0:00	/usr/libexec
amar	1535	0.0	0.1	314756	8848	?	Sl	00:41	0:00	/usr/libexec

kill - This command **terminates the running process** on a Linux machine.

To use these utilities you need to know the PID (process id) of the process you want to kill.

pidof - To find the PID of a process

In the below command, i am trying to get the process id of chrome

```
amar@amarkr:~/Desktop$ pidof chrome
```

```
9247 7826 7813 7759 7303 7258 7242 7204 7189 7144 7088 6376 6324 6292 6265 6239
6225 6181 6136 6112 6044 6023 6010 5997 5991 5957 5951 5928 5912 5884 5871 584
3 5493 4207 3825 3723 3704 3596 3411 3245 3238 3181 3174 3169 3154 3138 3108 30
94 3090 3069 3064 3063 3053
```

nice - Linux can run a lot of processes at a time, which can slow down the speed of some high priority processes and result in poor performance.

To avoid this, you can tell your machine to prioritize processes as per your requirements.

The Priority is called ***nice***ness in linux, and it has a value between -20 to 19. The lower the niceness index, the higher would be a priority given to that task.

The default value of all the processes is 0.

df - This utility reports the free disk space (Hard Disk) on all the file systems.

```
amar@amarkr:~/Desktop$ df
Filesystem      1K-blocks    Used Available Use% Mounted on
udev            3984388         0   3984388  0% /dev
tmpfs           802788      2192    800596  1% /run
/dev/sda6       102684472 14721324  82703888 16% /
tmpfs           4013932    179976   3833956  5% /dev/shm
tmpfs           5120         4        5116  1% /run/lock
tmpfs           4013932         0   4013932  0% /sys/fs/cgroup
/dev/loop0      56832     56832         0 100% /snap/core18/1988
/dev/loop1      56832     56832         0 100% /snap/core18/1997
/dev/loop2      223232    223232         0 100% /snap/gnome-3-34-1804/60
/dev/loop4      63616     63616         0 100% /snap/gtk-common-themes/1506
/dev/loop6      51072     51072         0 100% /snap/snap-store/467
/dev/loop7      52352     52352         0 100% /snap/snap-store/518
/dev/loop3      224256    224256         0 100% /snap/gnome-3-34-1804/66
/dev/loop5      66432     66432         0 100% /snap/gtk-common-themes/1514
/dev/loop9      33152     33152         0 100% /snap/snapd/11402
/dev/loop8      33152     33152         0 100% /snap/snapd/11588
/dev/loop10     20096     20096         0 100% /snap/video-downloader/749
/dev/loop11     20096     20096         0 100% /snap/video-downloader/756
/dev/sda1       262144     37000    225144 15% /boot/efi
tmpfs           802784         20    802764  1% /run/user/125
tmpfs           802784         40    802744  1% /run/user/1000
```

If you want the above information in readable format, then use the command

```
amar@amarkr:~/Desktop$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            3.8G   0    3.8G   0% /dev
tmpfs           784M  2.2M  782M   1% /run
/dev/sda6       98G   15G   79G   16% /
tmpfs           3.9G  170M  3.7G   5% /dev/shm
tmpfs           5.0M   4.0K  5.0M   1% /run/lock
tmpfs           3.9G   0    3.9G   0% /sys/fs/cgroup
/dev/loop0       56M   56M    0 100% /snap/core18/1988
/dev/loop1       56M   56M    0 100% /snap/core18/1997
/dev/loop2      218M  218M    0 100% /snap/gnome-3-34-1804/60
/dev/loop4       63M   63M    0 100% /snap/gtk-common-themes/1506
/dev/loop6       50M   50M    0 100% /snap/snap-store/467
/dev/loop7       52M   52M    0 100% /snap/snap-store/518
/dev/loop3      219M  219M    0 100% /snap/gnome-3-34-1804/66
/dev/loop5       65M   65M    0 100% /snap/gtk-common-themes/1514
/dev/loop9       33M   33M    0 100% /snap/snapd/11402
/dev/loop8       33M   33M    0 100% /snap/snapd/11588
/dev/loop10      20M   20M    0 100% /snap/video-downloader/749
/dev/loop11      20M   20M    0 100% /snap/video-downloader/756
/dev/sda1       256M   37M  220M  15% /boot/efi
tmpfs           784M   20K  784M   1% /run/user/125
tmpfs           784M   40K  784M   1% /run/user/1000
```

free - This command shows the free and used memory (RAM) on the Linux System.

```
amar@amarkr:~/Desktop$ free
              total        used          free      shared  buff/cache   available
Mem:           8027868       3184376       2971852        192880        1871640       4369848
Swap:          2097148           0         2097148
```

Linux Logging Basics

Operating system logs provide a wealth of diagnostic information about your computer, and linux is no exception. Everything from kernel events to user actions are logged by Linux, allowing you to see almost any action performed on your servers. In this section, we will explain what Linux logs are and where you can find them, and how to interpret them.

Linux System Logs

Linux has a special directory for storing logs called `./var/log`. This directory contains logs from the OS itself, services, and various applications running on the system. **Here's what this directory looks like on UBUNTU**

```
amar@amarkr:/var/log$ ls
alternatives.log      cups                  kern.log.2.gz
alternatives.log.1    dist-upgrade         kern.log.3.gz
alternatives.log.2.gz dmesg                kern.log.4.gz
alternatives.log.3.gz dmesg.0              lastlog
appport.log           dmesg.1.gz           openvpn
appport.log.1         dmesg.2.gz           private
appport.log.2.gz      dmesg.3.gz           speech-dispatcher
apt                   dmesg.4.gz           syslog
auth.log              dpkg.log             syslog.1
auth.log.1            dpkg.log.1           syslog.2.gz
auth.log.2.gz         dpkg.log.2.gz        syslog.3.gz
auth.log.3.gz         dpkg.log.3.gz        syslog.4.gz
auth.log.4.gz         faillog              syslog.5.gz
boot.log              fontconfig.log        syslog.6.gz
boot.log.1            gdm3                 ubuntu-advantage.log
boot.log.2            gpu-manager.log       unattended-upgrades
boot.log.3            gpu-manager-switch.log wtmp
boot.log.4            hp                   Xorg.0.log
boot.log.5            installer            Xorg.0.log.old
bootstrap.log         journal              Xorg.1.log
btmtp                 kern.log              Xorg.1.log.old
btmtp.1               kern.log.1
```

What is syslog ?

Syslog is a standard for creating and transmitting logs. The word “**syslog**” can refer to any of the following.

1. The syslog service, which receives and processes syslog messages. It listens for events by creating a socket located `/dev/log`, which applications can write to. It can write messages to a local file forward messages to a remote server. There are different syslog implementations including ***rsyslogd*** and ***syslog-ng***.
2. The syslog protocol (***RFC 5424***), which is a transport protocol that specifies how to transmit logs over a network. It is also a data format defining how messages are structured. By default, it uses port 514 for plaintext messages and port 6514 for encrypted messages.
3. A syslog message, which is any log formatted in the syslog message format. A syslog message consists of a standardized header and message containing the log’s contents.

Logging with systemd

Many Linux distributions ship with systemd, which is a process and service manager. Systemd implements its own logging service called journald that can replace or complement syslog. Journald logs in a significantly different manner than systemd, which is why it has its own section in the ultimate guide to logging.

Analyzing Linux Logs

There's a great deal of information stored within your Linux Logs, but the challenge is knowing how to extract it. There are a number of tools you can use to do this, from command line tools to more advanced analytics tools capable of searching on specific fields, calculating summaries, generating charts, and much more.

Searching with grep

One of the simplest ways to analyze logs is by performing plain text searches using grep. Grep is a command line tool that can search for matching text in a file, or in output from other commands. It's included by default in most Linux distributions and is also available for Windows and Mac.

To perform a simple search, enter your search string followed by the file you want to search. Here, we search the mergeSort.cpp file for lines containing "merge".

```
amar@amarkr:~/Desktop$ grep "merge" mergeSort.cpp
void merge(int *a, int beg, int end){
void mergeSort(int *a, int beg, int end){
    mergeSort(a,beg,mid);
    mergeSort(a,mid+1,end);
    merge(a,beg,end);
    mergeSort(a,0,argc-2);
```

This returns lines containing the exact match.

tail - Tail is another command line tool that can display the latest changes from a file in real time. This is useful for monitoring ongoing processes, such as restarting a service or testing a code change. You can also use tail to print the last few lines of a file, or pair it with grep to filter the output from a log file.

```
amar@amarkr:~/Desktop$ tail mergeSort.cpp
    a[i]=atoi(argv[i+1]);
}
mergeSort(a,0,argc-2);
for(int i=0 ; i<argc-1 ; ++i){
    cout<<a[i]<<" ";
}
cout<<endl;
cout<<"reached end"<<endl;
return 0;
}amar@amarkr:~/Desktop$
```

Filtering and Parsing with Awk

Filtering allows you to search on a specific field value instead of doing a full text search. This makes your log analysis more accurate because it will ignore undesired matches from other parts of the log message. In order to search on a field value, you need to parse your logs first, or at least have a way of searching based on the event structure. To do this, we can use awk.

Awk is a powerful command line tool that provides a complete scripting language, so you can filter and parse out fields more effectively.

Troubleshooting with Linux Logs

Troubleshooting is one of the main reasons people create logs. When a problem occurs, you'll want to diagnose it to understand why it happened and what the cause was. An error message or a sequence of events can give you clues to the root cause, indicate how to reproduce the problem, and guide you towards the solutions.

Login Failures

For security reasons, you may want to know which users have logged in or attempted to log in to your system. You can check your authentication logs for failed attempts, which occur when users provide incorrect credentials or don't have permission to login. This often occurs when using SSH for remote access or when using the **su** command to run a command as another user.

These types of authentication events are logged by the **pluggable authentication module** (PAM). Failed events often contain strings like "Failed password" and "user unknown", while successful authentication events often contain strings like "Accepted password" and "session opened".

Causes of Reboots

Sometimes a server can stop due to a system crash or reboot. To know how or who has done this you can use the shutdown command.

Kernel Initializing

If you want to see when the server restarted regardless of reason (including crashes), you can search the kernel log file which is located at **`/var/log/kern.log`**. Syslog also applies the “kern” facility to kernel logs. The following logs were generated immediately after boot. Note the timestamp between the brackets is 0: this tracks the amount of time since the kernel started.

Detect Memory Problems

There are several reasons a server might crash, but one common cause is running out of memory.

When **RAM** and **swap** space are completely exhausted, the kernel will start killing processes - typically those using the most memory and the most short-lived. The error occurs when your system is using all of its memory.

The error occurs when your system is using all of its memory, and a new or existing process attempts to access additional memory. Look in your log files for strings like “Out of Memory” or for kernel warnings. These strings indicate your system intentionally killed the process or application rather than allowing the process to crash.

Logging Cron Job Errors

The cron daemon is a scheduler that runs commands at specified dates and times. If the process fails to run or fails to finish, then a cron error appears in your log files. You can find these files in **`/var/log/cron`**, **`/var/log/messages`**, and **`/var/log/syslog`** depending on your distribution. To know more about cron job please visit: <https://en.wikipedia.org/wiki/Cron>

Managing Linux Logs

A key best practice for logging is to centralize or aggregate your logs in a single location, especially if you have multiple servers or architecture tiers. Modern applications often have multiple tiers of infrastructure that can include a mix of on-premise servers and cloud services. Trying to hunt down the right file to troubleshoot and error would be incredibly difficult, and trying to correlate problems across systems would be even harder. To understand more about this please go through this:

<https://www.loggly.com/ultimate-guide/managing-linux-logs/>



REFERENCES

Shell Scripting: https://www.tutorialspoint.com/unix/shell_scripting.htm

Linux System Administration Book: <https://linux-training.be/linuxsys.pdf>

Wikipedia: https://en.wikipedia.org/wiki/System_administrator