DevOps external course

Linux Processes

Lecture 5.4

Module 5 **Linux Essentials**

Serge Prykhodchenko

Q/A ON PREVIOUS

Types of Linux File Systems

A standard Linux Distribution provides the choice of partitioning disk with the file formats listed below, each of which has special meaning associated with it.

1. ext2
2. ext3
3. ext4
4. jfs
5. ReiserFS
6. XFS
7. Btrfs

Types of Linux File Systems

ext2, ext3, ext4

These are the progressive version of **Extended Filesystem** (**ext**), which primarily was developed for **MINIX**. The second extended version (**ext2**) was an improved version. **Ext3** added performance improvement. **Ext4** was a performance improvement besides additional providing additional features.

Read Also:

JFS

The **Journaled File System** (**JFS**) was developed by IBM for AIX UNIX which was used as an alternative to system ext. JFS is an alternative to **ext4** currently and is used where stability is required with the use of very few resources. When CPU power is limited JFS comes handy.

ReiserFS

It was introduced as an alternative to **ext3** with improved performance and advanced features. There was a time when **SuSE Linux**'s default file format was **ReiserFS** but later Reiser went out of business and SuSe had no option other than to return back to **ext3**. ReiserFS supports file System Extension dynamically which was relatively an advanced feature but the file system lacked certain area of performance.

XFS

XFS was a high speed **JFS** which aimed at parallel **I/O** processing. NASA still usages this file system on their **300+** terabyte storage server.

Btrfs

B-Tree File System (**Btrfs**) focus on fault tolerance, fun administration, repair System, large storage configuration and is still under development. Btrfs is not recommended for Production System.

Types of Linux File Systems

A **clustered file system** is a file system which is shared by being simultaneously mounted on multiple servers. There are several approaches to clustering, most of which do not employ a clustered file system (only direct attached storage for each node). Clustered file systems can provide features like location-independent addressing and redundancy which improve reliability or reduce the complexity of the other parts of the cluster. Parallel file systems are a type of clustered file system that spread data across multiple storage nodes, usually for redundancy or performance

A shared-disk file system uses a storage area network (SAN) to allow multiple computers to gain direct disk access at the block level. Access control and translation from file-level operations that applications use to block-level operations used by the SAN must take place on the client node. The most common type of clustered file system, the shared-disk file system —by adding mechanisms for concurrency control—provides a consistent and serializable view of the file system, avoiding corruption and unintended data loss even when multiple clients try to access the same files at the same time. Shared-disk file-systems commonly employ some sort of fencing mechanism to prevent data corruption in case of node failures, because an unfenced device can cause data corruption if it loses communication with its sister nodes and tries to access the same information other nodes are accessing.

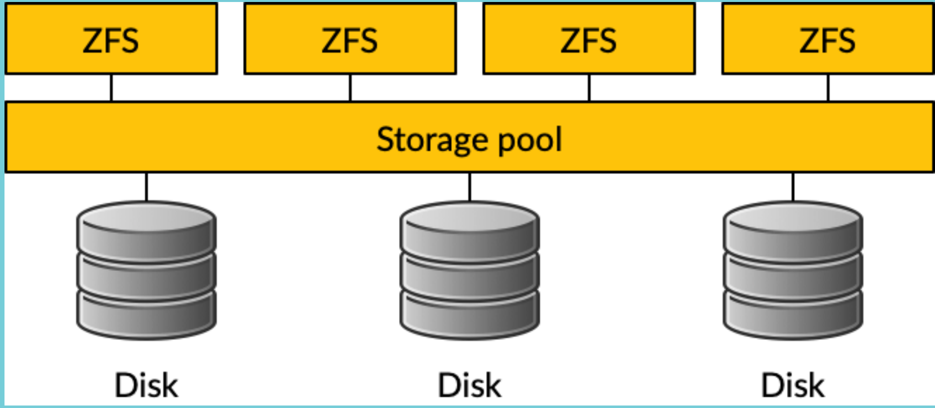
Distributed file systems do not share block level access to the same storage but use a network protocol.[3][4] These are commonly known as network file systems, even though they are not the only file systems that use the network to send data.[5] Distributed file systems can restrict access to the file system depending on access lists or capabilities on both the servers and the clients, depending on how the protocol is designed.

The difference between a distributed file system and a distributed data store is that a distributed file system allows files to be accessed using the same interfaces and semantics as local files – for example, mounting/unmounting, listing directories, read/write at byte boundaries, system's native permission model. Distributed data stores, by contrast, require using a different API or library and have different semantics (most often those of a database).

https://en.wikipedia.org/wiki/Clustered_file_system

ZFS pooled storage

https://nasa.cs.nctu.edu.tw/sa/2019/



- No partitions required
- Storage pool grows automatically
- All I/O bandwidth is always available
- All storage in the pool is shared

tar parameters

The Linux ‘tar’ stands for tape archive, is used to create Archive and extract the Archive files. tar command in Linux is one of the important command which provides archiving functionality in Linux.

Syntax:

tar [options] [archive-file] [file or directory to be archived]

Options:

-c : Creates Archive

-x : Extract the archive

-f : creates archive with given filename

-t : displays or lists files in archived file

-u : archives and adds to an existing archive file

-v : Displays Verbose Information

-A : Concatenates the archive files

-z : zip, tells tar command that creates tar file using gzip

-j : filter archive tar file using tbzip

-W : Verify a archive file

-r : update or add file or directory in already existed .tar file

tar parameters

1. **Creating an uncompressed tar Archive using option -cvf :** This command creates a tar file called file.tar which is the Archive of all .c files in current directory.
2. **Extracting files from Archive using option -xvf :** This command extracts files from Archives.
3. **gzip compression on the tar Archive, using option -z :** This command creates a tar file called file.tar.gz which is the Archive of .c files
4. **Extracting a gzip tar Archive *.tar.gz using option -xvzf :** This command extracts files from tar archived file.tar.gz files.
5. **Creating compressed tar archive file in Linux using option -j :** This command compresses and creates archive file less than the size of the gzip. Both compress and decompress takes more time then gzip.
6. **Untar single tar file or specified directory in Linux :** This command will Untar a file in current directory or in a specified directory using -C option
7. **Untar multiple .tar, .tar.gz, .tar.tbz file in Linux :** This command will extract or untar multiple files from the tar, tar.gz and tar.bz2 archive file. For example the above command will extract “fileA” “fileB” from the archive files
8. **Check size of existing tar, tar.gz, tar.tbz file in Linux :** The above command will display the size of archive file in Kilobytes(KB)
9. **Update existing tar file in Linux** $ tar rvf file.tar *.c

INTERACTIVE TEXT PROCESSING

vi

What is vi?

The default editor that comes with the UNIX operating system is called vi (visual editor).

The UNIX vi editor is a full screen editor and has two modes of operation:

Command mode commands which cause action to be taken on the file, and

Insert mode in which entered text is inserted into the file.

In the command mode, every character typed is a command that does something to the text file being edited; a character typed in the command mode may even cause the vi editor to enter the insert mode. In the insert mode, every character typed is added to the text in the file; pressing the <Esc> (Escape) key turns off the Insert mode.

While there are a number of vi commands, just a handful of these is usually sufficient for beginning vi users. To assist such users, this Web page contains a sampling of basic vi commands. The most basic and useful commands are marked with an asterisk (* or star) in the tables below. With practice, these commands should become automatic.

NOTE: Both UNIX and vi are case-sensitive. Be sure not to use a capital letter in place of a lowercase letter; the results will not be what you expect.

https://www.cs.colostate.edu/helpdocs/vi.html

To Get Into and Out Of vi

To Start vi

To use vi on a file, type in vi filename. If the file named filename exists, then the first page (or screen) of the file will be displayed; if the file does not exist, then an empty file and screen are created into which you may enter text.

*	vi filename	*edit filename starting at line 1*
	vi -r filename	*recover filename that was being edited when system crashed*

To Exit vi

Usually the new or modified file is saved when you leave vi. However, it is also possible to quit vi without saving the file.

Note: The cursor moves to bottom of screen whenever a colon (:) is typed. This type of command is completed by hitting the <Return> (or <Enter>) key.

*	:x<Return>	*quit vi, writing out modified file to file named in original invocation*
	:wq<Return>	*quit vi, writing out modified file to file named in original invocation*
	:q<Return>	*quit (or exit) vi*
*	:q!<Return>	*quit vi even though latest changes have not been saved for this vi call*

vi: Moving the Cursor

Unlike many of the PC and MacIntosh editors, the mouse does not move the cursor within the vi editor screen (or window). You must use the key commands listed below. On some UNIX platforms, the arrow keys may be used as well; however, since vi was designed with the Qwerty keyboard (containing no arrow keys) in mind, the arrow keys sometimes produce strange effects in vi and should be avoided.

If you go back and forth between a PC environment and a UNIX environment, you may find that this dissimilarity in methods for cursor movement is the most frustrating difference between the two.

In the table below, the symbol ^ before a letter means that the <Ctrl> key should be held down while the letter key is pressed.

*	j *or* <Return> [*or* down-arrow]	*move cursor down one line*
*	k [*or* up-arrow]	*move cursor up one line*
*	h *or* <Backspace> [*or* left-arrow]	*move cursor left one character*
*	l *or* <Space> [*or* right-arrow]	*move cursor right one character*
*	0 (zero)	*move cursor to start of current line (the one with the cursor)*
*	$	*move cursor to end of current line*
	w	*move cursor to beginning of next word*
	b	*move cursor back to beginning of preceding word*
	:0<Return> *or* 1G	*move cursor to first line in file*
	:n<Return> *or* nG	*move cursor to line n*
	:$<Return> *or* G	*move cursor to last line in file*

vi: Screen Manipulation

The following commands allow the vi editor screen (or window) to move up or down several lines and to be refreshed.

^f	*move forward one screen*
^b	*move backward one screen*
^d	*move down (forward) one half screen*
^u	*move up (back) one half screen*
^l	*redraws the screen*
^r	*redraws the screen, removing deleted lines*

vi: Inserting or Adding Text

The following commands allow you to insert and add text. Each of these commands puts the vi editor into insert mode; thus, the <Esc> key must be pressed to terminate the entry of text and to put the vi editor back into command mode.

*	i	*insert text before cursor, until <Esc> hit*
	I	*insert text at beginning of current line, until <Esc> hit*
*	a	*append text after cursor, until <Esc> hit*
	A	*append text to end of current line, until <Esc> hit*
*	o	*open and put text in a new line below current line, until <Esc> hit*
*	O	*open and put text in a new line above current line, until <Esc> hit*

Changing Text
The following commands allow you to modify text.

*	r	*replace single character under cursor (no <Esc> needed)*
	R	*replace characters, starting with current cursor position, until <Esc> hit*
	cw	*change the current word with new text, starting with the character under cursor, until <Esc> hit*
	cNw	*change N words beginning with character under cursor, until <Esc> hit; e.g., c5w changes 5 words*
	C	*change (replace) the characters in the current line, until <Esc> hit*
	cc	*change (replace) the entire current line, stopping when <Esc> is hit*
	Ncc *or* cNc	*change (replace) the next N lines, starting with the current line, stopping when <Esc> is hit*

vi: Deleting Text

The following commands allow you to delete text.

*	x	*delete single character under cursor*
	Nx	*delete N characters, starting with character under cursor*
	dw	*delete the single word beginning with character under cursor*
	dNw	*delete N words beginning with character under cursor; e.g., d5w deletes 5 words*
	D	*delete the remainder of the line, starting with current cursor position*
*	dd	*delete entire current line*
	Ndd *or* dNd	*delete N lines, beginning with the current line; e.g., 5dd deletes 5 lines*

Cutting and Pasting Text
The following commands allow you to copy and paste text.

yy	*copy (yank, cut) the current line into the buffer*
Nyy *or* yNy	*copy (yank, cut) the next N lines, including the current line, into the buffer*
p	*put (paste) the line(s) in the buffer into the text after the current line*

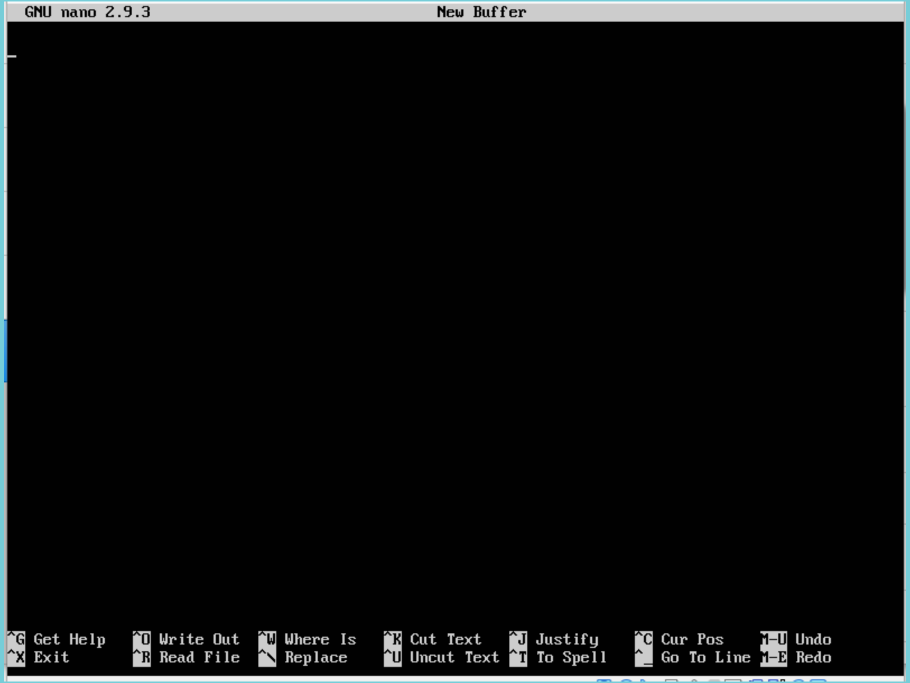
nano

- GNU nano is an easy to use command line text editor for Unix and Linux operating systems. It includes all the basic functionality you’d expect from a regular text editor, like syntax highlighting, multiple buffers, search and replace with regular expression support, spellchecking, UTF-8 encoding, and more.

https://linuxize.com/post/how-to-use-nano-text-editor/

nano

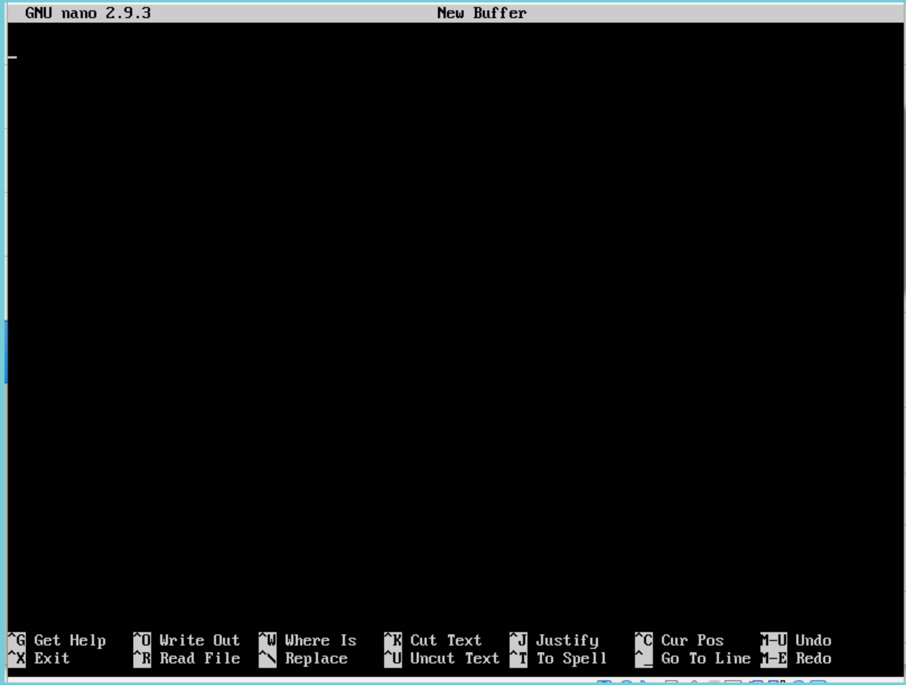
- To open an existing file or to create a new file, type nano followed by the file name:
- **$ nano filename**



```
GNU nano 2.9.3                              New Buffer

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    M-U Undo
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line M-E Redo
```

nano

- This opens a new editor window, and you can start editing the file.
- At the bottom of the window, there is a list of the most basic command shortcuts to use with the nano editor.
- All commands are prefixed with either ^ or M character. The caret symbol (^) represents the Ctrl key. For example, the ^J commands mean to press the Ctrl and J keys at the same time. The letter M represents the Alt key.



```
  GNU nano 2.9.3                    New Buffer

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    M-U Undo
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line M-E Redo
```

nano

- **Editing Files**
- To move the cursor to a specific line and character number, use the Ctrl+_ command. The menu on the bottom of the screen will change. Enter the number(s) in the "Enter line number, column number:" field and hit Enter.
- **Searching and replacing**
- To search for a text, press Ctrl+w, type in the search term, and press Enter. The cursor will move to the first match. To move to the next match, press Alt+w.
- If you want to search and replace, press Ctrl+\. Enter the search term and the text to be replaced with. The editor will move to the first match and ask you whether to replace it. After hitting Y or N it will move to the next match. Pressing A will replace all matches.

nano

- **Copping, cutting, and pasting**
- To select text, move the cursor to the beginning of the text and press Alt+a. This will set a selection mark. Move the cursor to the end of the text you want to select using the arrow keys. The selected text will be highlighted. If you want to cancel the selection press Ctrl+6
- Copy the selected text to the clipboard using the Alt+6 command. Ctrl+k will cut the selected text.
- If you want to cut whole lines, simply move the cursor to the line and press Ctrl+k. You can cut multiple lines by hitting Ctrl+k several times.

nano

- **Saving and Exiting**
- To save the changes you’ve made to the file, press Ctrl+o. If the file doesn’t already exist, it will be created once you save it.
- To exit nano press Ctrl+x. If there are unsaved changes, you’ll be asked whether you want to save the changes.
- To save the file, you must have at write permissions to the file. If you are creating a new file , you need to have write permission to the directory where the file is created.

WHERE IS DATA STORED

logger

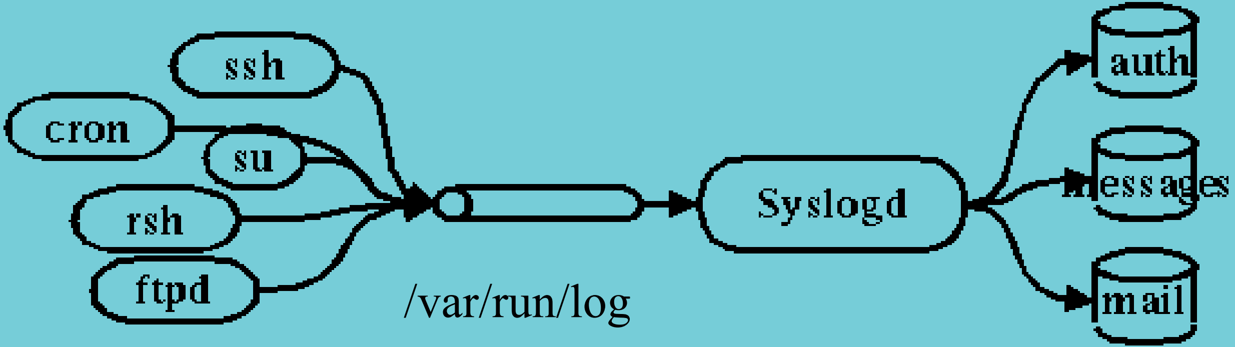
logger is a unix utility that provides a command interface to the syslog syslog module.
Parameters
-i - Log the id of the process being logged with each line.
-s -Put message not only to the log, but also to the standard error device.-f file - Log from the given file.
-p priority - Enter the message with the given priority.

The priority can be specified numerically or as a pair of facility.level. For example, -p local3.info logs the message as info level in the local3 facility. The default is user.notice.-t tag - Mark each line in the log with the specified tag.Message - Writes a message to the log; if it is not specified and the -f flag is not specified, then logging is done from standard input.The logger utility returns 0 on success and> 0 on errors.

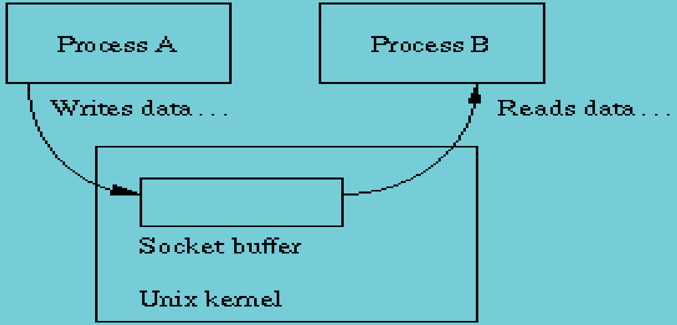
Syslog – The system event logger (1)

- Two main functions
  - To release programmers from the tedious of writing log files
  - To put administrators in control of logging
- Three parts:
  - syslogd, /etc/syslog.conf
    - The logging daemon and configure file
  - openlog(), syslog(), closelog()
    - Library routines to use syslogd
  - logger
    - A user command that use syslogd from shell

Syslog –
The system event logger (2)



```
ls -al /var/run/log /var/run/logpriv /dev/klog
crw-------  1 root  wheel  0x17 Sep  9 18:19 /dev/klog
srw-rw-rw-  1 root  wheel     0 Sep  9 18:20 /var/run/log
srw-------  1 root  wheel     0 Sep  9 18:20 /var/run/logpriv
```



/var/log - where most log files are stored

The system logging function (so-called "logs" or logging) is the main source of information about system operation and errors.
Logging can be performed on the local system, and log messages can be sent to a remote system, in addition, in the configuration file /etc/syslog.conf (in some new distributions replaced by /etc/rsyslog.conf), you can fine-tune the logging level. Logging is done using the syslogd daemon (rsyslogd on some newer distributions), which usually receives input from the /dev/log socket (locally) or from udp port 514 (from remote machines).

```
syslog-server:~# ls -l /dev/log
srw-rw-rw- 1 root root 0 Дек 17 06:25 /dev/log
```

/var/log - where most log files are stored

Logging is the main source of information about system operation and errors. This quick guide will cover the basic aspects of operating system logging, directory structure, programs for reading and viewing logs.

Main log files
All log files can be classified into one of the following categories:

- applications;
- developments;
- service;
- system

/var/log - where most log files are stored

Most of the log files are contained in the /var/log directory.
/var/log/syslog or **/var/log/messages** contains the global system log, which records messages from the moment the system was started, from the Linux kernel, various services, discovered devices, network interfaces, and much more.
/var/log/auth.log or **/var/log/secure** - information about user authorization, including successful and unsuccessful login attempts, as well as the authentication mechanisms involved.
/var/log/dmesg - device drivers. With the command of the same name, you can view the output of the file contents. The size of the log is limited, when the file reaches its limit, older messages will be overwritten with newer ones. By setting the key --level =, you can filter the output by the criterion of importance.

/var/log - where most log files are stored

Supported log levels (priorities):

- emerg - unused system
- alert - action should be taken immediately
- crit - criticality conditions
- err - error conditions
- warn - warning conditions
- notice - common but significant conditions
- info - informational
- debug - debug messages

```
(5:520)$ dmesg -l err
[1131424.604352] usb 1-1.1: 2:1: cannot get freq at ep 0x1
[1131424.666013] usb 1-1.1: 1:1: cannot get freq at ep 0x81
[1131424.749378] usb 1-1.1: 1:1: cannot get freq at ep 0x81
```

/var/log - where most log files are stored

/var/log/alternatives.log - The output of the update-alternatives program, which contains symbolic links to the default commands or libraries.
/var/log/anaconda.log - Logs logged during system installation.
/var/log/audit - Records generated by auditd.
/var/log/boot.log - Information that is written when the operating system boots.
/var/log/cron - The crond service report on the commands being executed and messages from the commands themselves.
/var/log/cups - Everything related to printing and printers.

/var/log - where most log files are stored

/ var / log / faillog - Failed login attempts. Very useful for checking security threats, hacker attacks, brute-force attacks. You can read the content using the faillog command.
/var / log / kern.log - The log contains messages from the kernel and warnings that can be useful in troubleshooting custom kernel modules.
/ var / log / maillog / or **/var/log/mail.log** - The log of the mail server used on the OS.
/var/log/pm-powersave.log - Messages from the battery saving service.
/ var / log / samba / - The logs of the Samba file server, which is used to access Windows shared folders and give Windows users access to Linux shared folders.
/ var / log / spooler - For old school people, contains USENET messages. Most often it is empty and abandoned.
/var/log/Xorg.0.log - X server logs. Most often they are useless, but if they have lines starting with EE, then you should pay attention to them.

/var/log - where most log files are stored

There will be a separate package manager log for each distribution.

/var/log/yum.log - For programs installed with Yum on RedHat Linux.
/var/log/emerge.log - For ebuilds installed from Portage via emerge on Gentoo Linux.
/var/log/dpkg.log - For programs installed with dpkg on Debian Linux and all related family of distributions.

/var/log - where most log files are stored

/ var / log / lastlog - Last user session. You can read it with the last command.
/ var / log / tallylog - Audit failed login attempts. Output to the screen using the pam_tally2 utility.
/ var / log / btmp - Every single log of failed login attempts. Just like that, just in case you haven't guessed yet where to look for traces of hackers' activity.
/ var / log / utmp - List of user logins to the system at the moment.
/ var / log / wtmp - Another log to record user logins. Output to the screen by utmpdump command.

```
(5:535)$ sudo utmpdump /var/log/wtmp
[8] [00788] [tty1] [        ] [tty1        ] [                    ] [0.0.0.0        ] [2020-04-15T16:59:32,524545+0000]
[1] [00000] [~~  ] [shutdown] [~           ] [4.15.0-20-generic   ] [0.0.0.0        ] [2020-04-15T16:59:33,596020+0000]
[2] [00000] [~~  ] [reboot  ] [~           ] [4.15.0-20-generic   ] [0.0.0.0        ] [2020-04-15T16:59:44,019201+0000]
[1] [00053] [~~  ] [runlevel] [~           ] [4.15.0-20-generic   ] [0.0.0.0        ] [2020-04-15T17:00:30,690729+0000]
[5] [00795] [tty1] [        ] [tty1        ] [                    ] [0.0.0.0        ] [2020-04-15T17:00:30,696481+0000]
[6] [00795] [tty1] [LOGIN   ] [tty1        ] [                    ] [0.0.0.0        ] [2020-04-15T17:00:30,696481+0000]
[7] [00822] [:0  ] [serge   ] [tty7        ] [:0                  ] [0.0.0.0        ] [2020-04-15T17:00:34,237698+0000]
```

/var/log - where most log files are stored

/ var / log / mysql / - MySQL database log.
/ var / log / httpd / or **/ var / log / apache2 /** - Apache web server log, access log is in access_log, and errors are in error_log.
/ var / log / lighthttpd / - Lighttpd web server log.

The user's home directory may contain graphical application logs, DE.
~ / .xsession-errors - Stderr output of X11 graphical applications.
~ / .xfce4-session.verbose-log - XFCE4 desktop messages.

Libraries

- Most binary applications make use of dynamic linking

- Some functionality is not in the application but is taken from 'library' files
  - Come with the operating system
  - Distributed with the application

- Libraries allow for
  - Updating without rebuilding the application
  - Removal of duplicate code

Checking library compatibility

- Most binaries are dynamically linked
  - Additional functionality pulled from system libraries at runtime
- If the libraries are missing / wrong versions then the application won’t work
- Installing the missing libraries will usually fix things.

Example of ldd usage

```
$ ldd `which ssh`
        linux-vdso.so.1 (0x00007ffef7598000)
        libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007f68e7899000)
        libcrypto.so.1.0.0 => /usr/lib/x86_64-linux-gnu/libcrypto.so.1.0.0 (0x00007f68e7456000)
        libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f68e7252000)
        libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007f68e7035000)
        libresolv.so.2 => /lib/x86_64-linux-gnu/libresolv.so.2 (0x00007f68e6e1a000)
        libgssapi_krb5.so.2 => /usr/lib/x86_64-linux-gnu/libgssapi_krb5.so.2 (0x00007f68e6bcf000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f68e67de000)
        libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007f68e656c000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f68e7d77000)
        libkrb5.so.3 => /usr/lib/x86_64-linux-gnu/libkrb5.so.3 (0x00007f68e6296000)
        libk5crypto.so.3 => /usr/lib/x86_64-linux-gnu/libk5crypto.so.3 (0x00007f68e6064000)
        libcom_err.so.2 => /lib/x86_64-linux-gnu/libcom_err.so.2 (0x00007f68e5e60000)
        libkrb5support.so.0 => /usr/lib/x86_64-linux-gnu/libkrb5support.so.0 (0x00007f68e5c55000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f68e5a36000)
        libkeyutils.so.1 => /lib/x86_64-linux-gnu/libkeyutils.so.1 (0x00007f68e5832000)
```

Library search path

- Libraries are identified by using a search path
  - Some hard-coded components
  - Some components added by the application
  - Some components can be configured by the user

- System library directories (`/lib /usr/lib`)
- Application library directories (usually relative to the application itself)
- User components (via the `LD_LIBRARY_PATH` environment variable)

PROCESS MANAGEMENT BASICS

Process Management Basics

Types of Processes

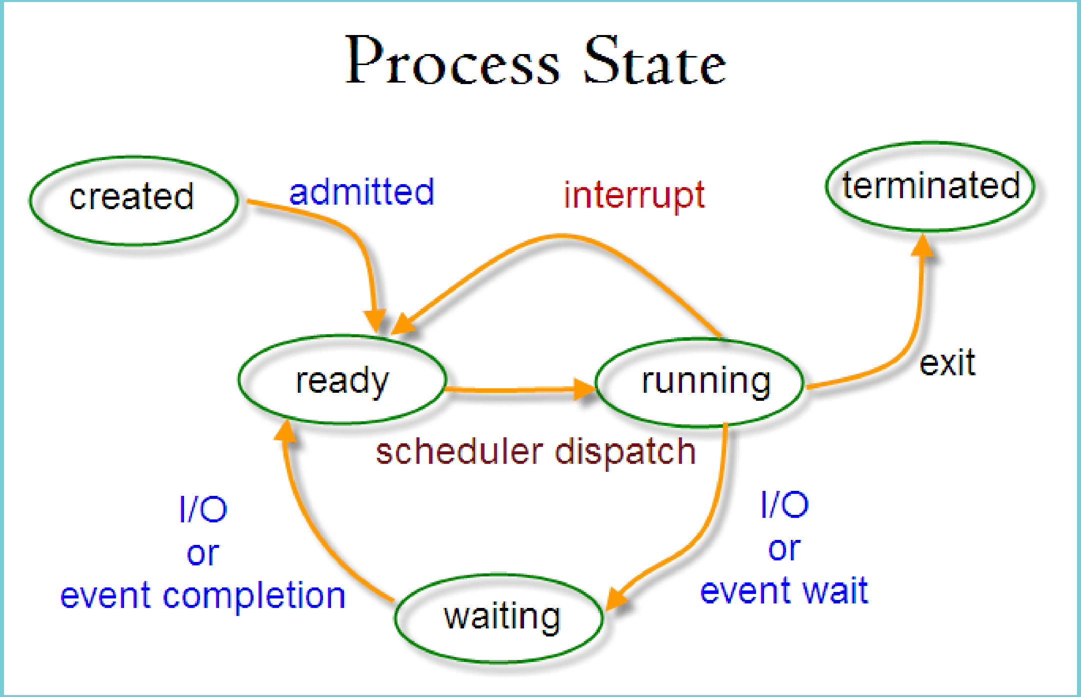
There are fundamentally two types of processes in Linux:

•***Foreground processes*** (also referred to as interactive processes) – these are initialized and controlled through a terminal session. In other words, there has to be a user connected to the system to start such processes; they haven't started automatically as part of the system functions/services.

•***Background processes*** (also referred to as non-interactive/automatic processes) – are processes not connected to a terminal; they don't expect any user input

Process Management Basics

These are special types of background processes that start at system startup and keep running forever as a service; they don’t die. They are started as system tasks (run as services). However, they can be controlled by a user via the **init** process.



Process Management Basics

For everything that happens on a Linux server, a process is started. For that reason, process management is among the key skills that an administrator has to master. To do this efficiently, it is important to know which type of process you are dealing with. A major distinction can be made between two process types:

- Shell jobs are commands started from the command line. They are associated with the shell that was current when the process was started. Shell jobs are also referred to as interactive processes.
- Daemons are processes that provide services. They normally are started when a computer is booted and often (but certainly not in all cases) they are running with root privileges.

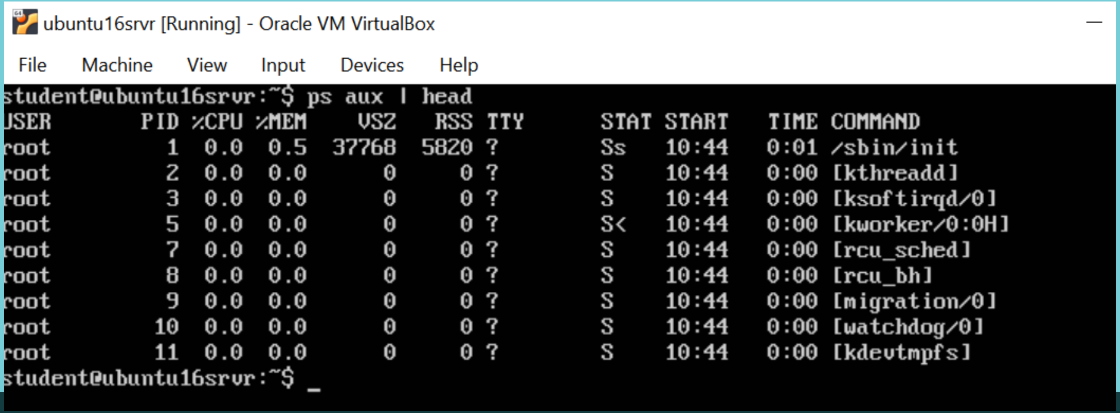
Process Management Basics

When a process is started, it can use multiple threads. A thread is a task started by a process and that a dedicated CPU can service. The Linux shell offers tools to manage individual threads. Thread management should be taken care of from within the command.

To manage a process efficiently, it is paramount that you know what type of process you are dealing with. Shell jobs require another approach than the processes that are automatically started when a computer boots.

Process Management Basics

There are two different types of background processes.
There are kernel threads. These are a part of the Linux kernel, and each of them is started with its own process identification number (PID). When managing processes, it is easy to recognize the kernel processes because they have a name that is between square brackets. Listing shows a list of a few processes as output of the command:
> ps aux | head
in which you can see a couple of kernel threads. As an administrator, it is important to know that kernel threads cannot be managed. You cannot adjust their priority; neither is it possible to kill them, except by taking the entire machine down.



```
student@ubuntu16srvr:~$ ps aux | head
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.5  37768  5820 ?        Ss   10:44   0:01 /sbin/init
root         2  0.0  0.0      0     0 ?        S    10:44   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    10:44   0:00 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   10:44   0:00 [kworker/0:0H]
root         7  0.0  0.0      0     0 ?        S    10:44   0:00 [rcu_sched]
root         8  0.0  0.0      0     0 ?        S    10:44   0:00 [rcu_bh]
root         9  0.0  0.0      0     0 ?        S    10:44   0:00 [migration/0]
root        10  0.0  0.0      0     0 ?        S    10:44   0:00 [watchdog/0]
root        11  0.0  0.0      0     0 ?        S    10:44   0:00 [kdevtmpfs]
student@ubuntu16srvr:~$ _
```

Process Management Basics

Using ***ps*** to Get Process Information

The most common command to get an overview of currently running processes is ***ps***.

If used without any arguments, the ***ps*** command shows only those processes that have been started by the current user. You can use many different options to display different process properties.

If you are looking for a short summary of the active processes, use ***ps aux***.

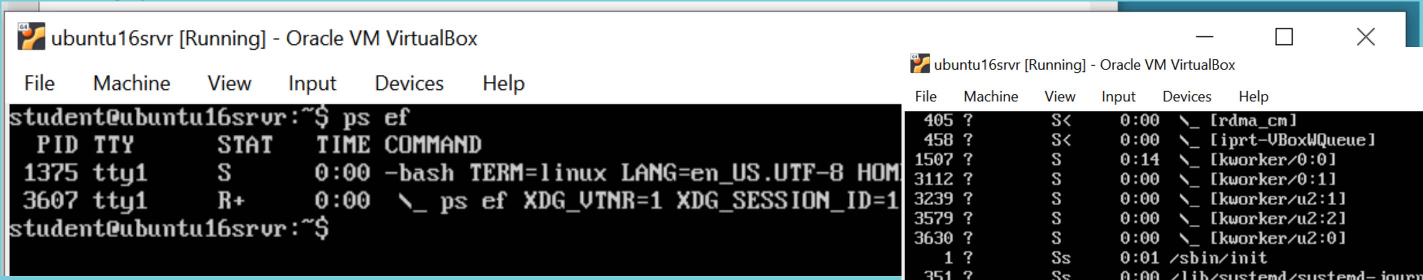
If you are not only looking for the name of the process but also for the exact command that was used to start the process, use ***ps -ef***.

Alternative ways to use ***ps exist*** as well, such as the command

ps fax , which shows hierarchical relationships between parent and child processes.

Process Management Basics

Using *ps* to Get Process Information

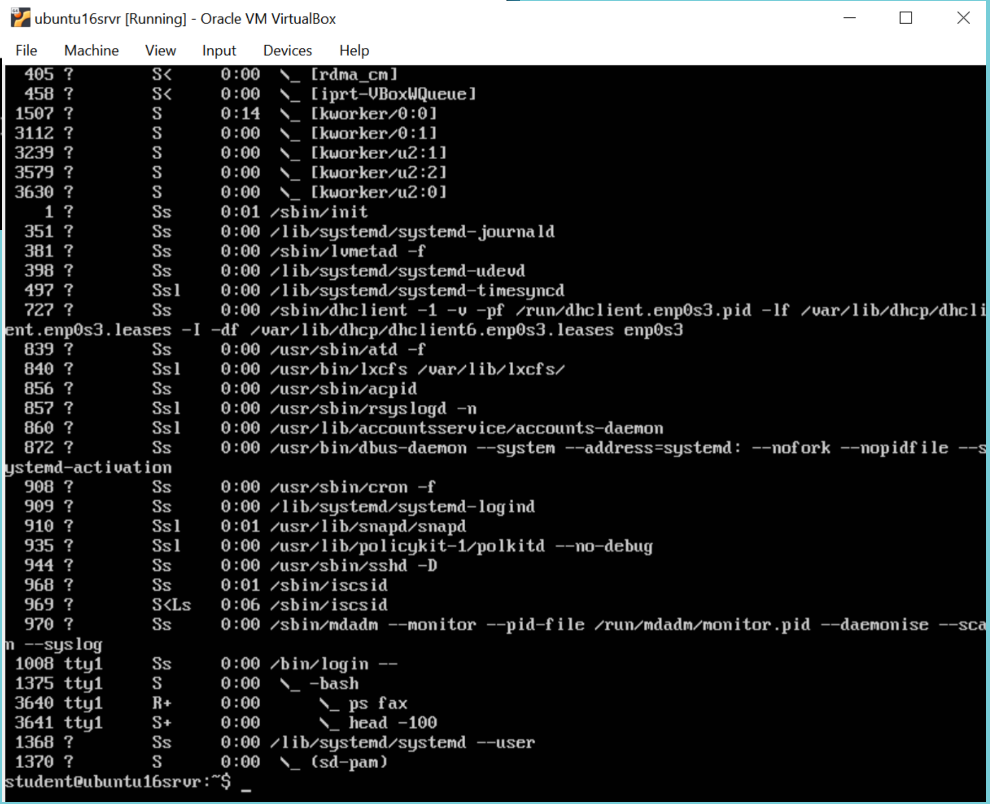


ubuntu16srvr [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

```
student@ubuntu16srvr:~$ ps ef
  PID TTY      STAT   TIME COMMAND
 1375 tty1     S      0:00 -bash TERM=linux LANG=en_US.UTF-8 HOM
 3607 tty1     R+     0:00  \_ ps ef XDG_VTNR=1 XDG_SESSION_ID=1
student@ubuntu16srvr:~$
```

For some commands, using a hyphen before options is optional. Some commands do not. The ***ps*** command is one of those latter commands. There is a historical reason why this is the case: The commands derive from the old BSD UNIX flavor, where it was common to specify command-line options without a - in front of them.



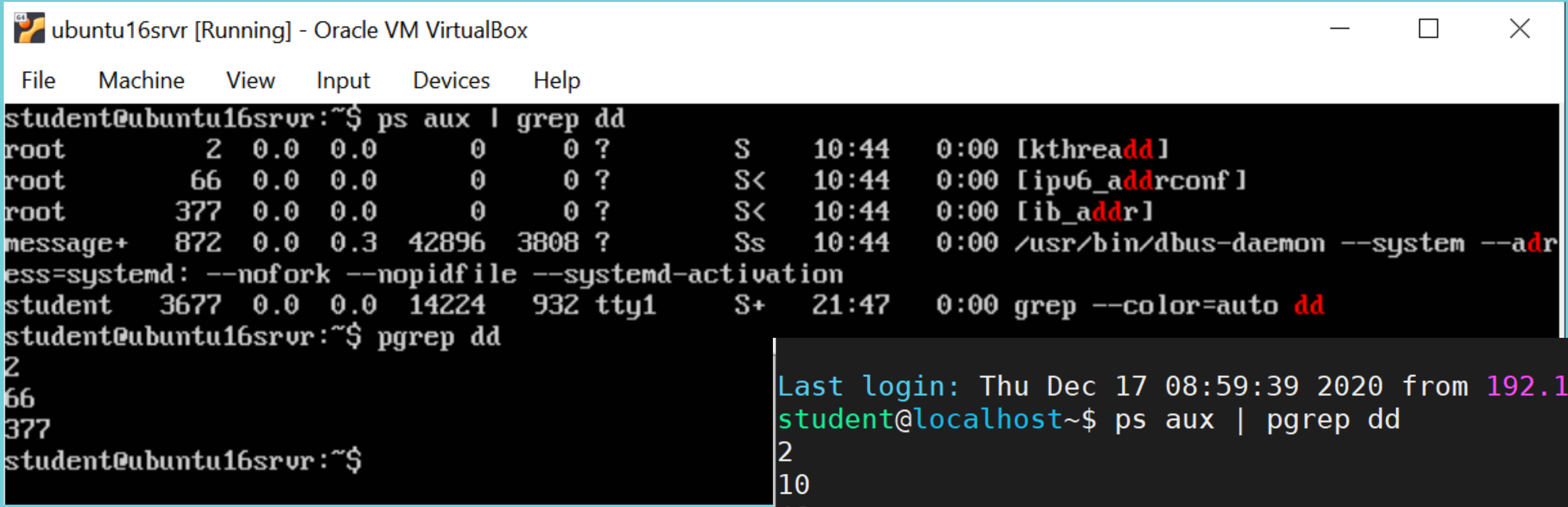
ubuntu16srvr [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

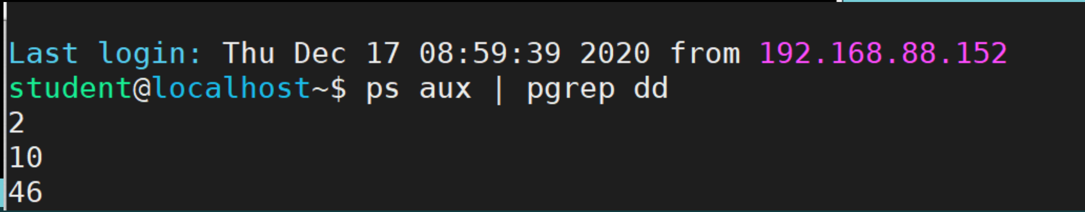
```
  405 ?        S<     0:00  \_ [rdma_cm]
  458 ?        S<     0:00  \_ [iprt-VBoxWQueue]
 1507 ?        S      0:14  \_ [kworker/0:0]
 3112 ?        S      0:00  \_ [kworker/0:1]
 3239 ?        S      0:00  \_ [kworker/u2:1]
 3579 ?        S      0:00  \_ [kworker/u2:2]
 3630 ?        S      0:00  \_ [kworker/u2:0]
    1 ?        Ss     0:01 /sbin/init
  351 ?        Ss     0:00 /lib/systemd/systemd-journald
  381 ?        Ss     0:00 /sbin/lvmetad -f
  398 ?        Ss     0:00 /lib/systemd/systemd-udevd
  497 ?        Ssl    0:00 /lib/systemd/systemd-timesyncd
  727 ?        Ss     0:00 /sbin/dhclient -1 -v -pf /run/dhclient.enp0s3.pid -lf /var/lib/dhcp/dhcli
ent.enp0s3.leases -I -df /var/lib/dhcp/dhclient6.enp0s3.leases enp0s3
  839 ?        Ss     0:00 /usr/sbin/atd -f
  840 ?        Ssl    0:00 /usr/bin/lxcfs /var/lib/lxcfs/
  856 ?        Ss     0:00 /usr/sbin/acpid
  857 ?        Ssl    0:00 /usr/sbin/rsyslogd -n
  860 ?        Ssl    0:00 /usr/lib/accountsservice/accounts-daemon
  872 ?        Ss     0:00 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --s
ystemd-activation
  908 ?        Ss     0:00 /usr/sbin/cron -f
  909 ?        Ss     0:00 /lib/systemd/systemd-logind
  910 ?        Ssl    0:01 /usr/lib/snapd/snapd
  935 ?        Ssl    0:00 /usr/lib/policykit-1/polkitd --no-debug
  944 ?        Ss     0:00 /usr/sbin/sshd -D
  968 ?        Ss     0:01 /sbin/iscsid
  969 ?        S<Ls   0:06 /sbin/iscsid
  970 ?        Ss     0:00 /sbin/mdadm --monitor --pid-file /run/mdadm/monitor.pid --daemonise --sca
n --syslog
 1008 tty1     Ss     0:00 /bin/login --
 1375 tty1     S      0:00  \_ -bash
 3640 tty1     R+     0:00      \_ ps fax
 3641 tty1     S+     0:00      \_ head -100
 1368 ?        Ss     0:00 /lib/systemd/systemd --user
 1370 ?        S      0:00  \_ (sd-pam)
student@ubuntu16srvr:~$ _
```

Process Management Basics

Using ***ps*** to Get Process Information
An important piece of information to get out of the ***ps*** command is the PID. Many tasks require the PID to operate, and that is why a command like ***ps aux | grep dd*** , which will show process details about dd, including its PID, is quite common. An alternative way to get the same result is to use the pgrep command. Use ***pgrep dd*** to get a list of all PIDs that have a name containing the string ***dd***.


ubuntu16srvr [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

```
student@ubuntu16srvr:~$ ps aux | grep dd
root         2  0.0  0.0      0     0 ?        S    10:44   0:00 [kthreadd]
root        66  0.0  0.0      0     0 ?        S<   10:44   0:00 [ipv6_addrconf]
root       377  0.0  0.0      0     0 ?        S<   10:44   0:00 [ib_addr]
message+   872  0.0  0.3  42896  3808 ?        Ss   10:44   0:00 /usr/bin/dbus-daemon --system --adr
ess=systemd: --nofork --nopidfile --systemd-activation
student   3677  0.0  0.0  14224   932 tty1     S+   21:47   0:00 grep --color=auto dd
student@ubuntu16srvr:~$ pgrep dd
2
66
377
student@ubuntu16srvr:~$
```



```
Last login: Thu Dec 17 08:59:39 2020 from 192.168.88.152
student@localhost~$ ps aux | pgrep dd
2
10
46
```

Process Management Basics

Adjusting Process Priority with ***nice***

When Linux processes are started, they are started with a specific priority. By default, all regular processes are equal and are started with the same priority, which is the priority number 20. In some cases, it is useful to change the default priority that was assigned to the process when it was started. You can do that using the ***nice*** and ***renice*** commands. Use ***nice*** if you want to start a process with an adjusted priority. Use ***renice*** to change the priority for a currently active process. Alternatively, you can use the ***r*** command from the ***top*** utility to change the priority of a currently running process.

When using ***nice*** or ***renice*** to adjust process priority, you can select from values ranging from -0 to 19. The default niceness of a process is set to 0 (which results in the priority value of 20). By applying a negative niceness, you increase the priority. Use a positive niceness to decrease the priority. It is a good idea not to use the ultimate values immediately. Instead, use increments of 5 and see how it affects the application.

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:

ping c5 www.tecmint.com &

Run two command in background, simultaneously:

apt-get update & apt-get upgrade &

Process Management Basics

Examples of using ***nice*** and ***renice***.

The command ***nice -n 5 dd if=/dev/zero of=/dev/null &*** starts an infinite I/O-intensive job, but with an adjusted niceness so that some place remains for other processes as well. To adjust the niceness of a currently running process, you need the PID of that process. The following two commands show how ***ps aux*** is used to find the PID of the ***dd*** job from the previous example. Next, you see how the***renice*** command is used to change the niceness of that command:

1. Use ***ps aux | grep dd*** to find the PID of the dd command that you just started. The PID is in the second column of the command output.
2. Use ***renice -n 10 -p*** XXYY (assuming that XXYY is the PID you just found).

Note that regular users can only decrease the priority of a running process.

You must be root to give processes increased priority.

Process Management Basics

Sending Signals to Processes with ***kill***, ***killall***, and ***pkill***

Before starting to think about using the ***kill*** command or sending other signals to processes, it is good to know that Linux processes have a hierarchical relationship. Every process has a parent process, and as long as it lives, the parent process is responsible for the child processes it has created. This is particularly important when processes are terminated, because it is the parent that has to clean up the resources that were used by the children. When using ***kill*** on a parent process that still has active children, you will for that reason not just kill the parent process in question but also all of its currently active child processes.

The Linux kernel allows many signals to be sent to processes. Use ***man*** for a complete overview of all the available signals. Three of these signals work for all processes:

The signal SIGTERM (15) is used to ask a process to stop.

The signal SIGKILL (9) is used to force a process to stop.

The SIGHUP (1) signal is used to hang up a process. The effect is that the process will reread its configuration files, which makes this a useful signal to use after making modifications to a process configuration file.

Process Management Basics

To send a signal to a process, the ***kill*** command is used. The most common use is the need to stop a process, which you can do by using the ***kill*** command followed by the PID of the process. This sends the ***SIGTERM*** signal to the process, which normally causes the process to cease its activity.

Sometimes the ***kill*** command does not work because the process you want to kill is busy. In that case, you can use ***kill -9*** to send the ***SIGKILL*** signal to the process. Because the ***SIGKILL*** signal cannot be ignored, it forces the process to stop, but you also risk losing data while using this command. In general, it is a bad idea to use ***kill -9*** :

- You risk losing data.
- Your system may become unstable if other processes depend on the process you have just killed.

Use ***kill -l*** to show a list of available signals that can be used with ***kill*** .

Process Management Basics

There are some commands that are related to kill: ***killall*** and ***pkill*** . The ***pkill*** command is a bit easier to use because it takes the name rather than the PID of the process as an argument. You can use the ***killall*** command if multiple processes using the same name need to be killed simultaneously.

Using ***killall*** was particularly common when Linux environments were multiprocessing instead of multithreading. In a multiprocessing environment where a server starts several commands, all with the same name, it is not easy to stop these commands one by one based on their individual PID.

Using ***killall*** enables you to terminate all these processes simultaneously.

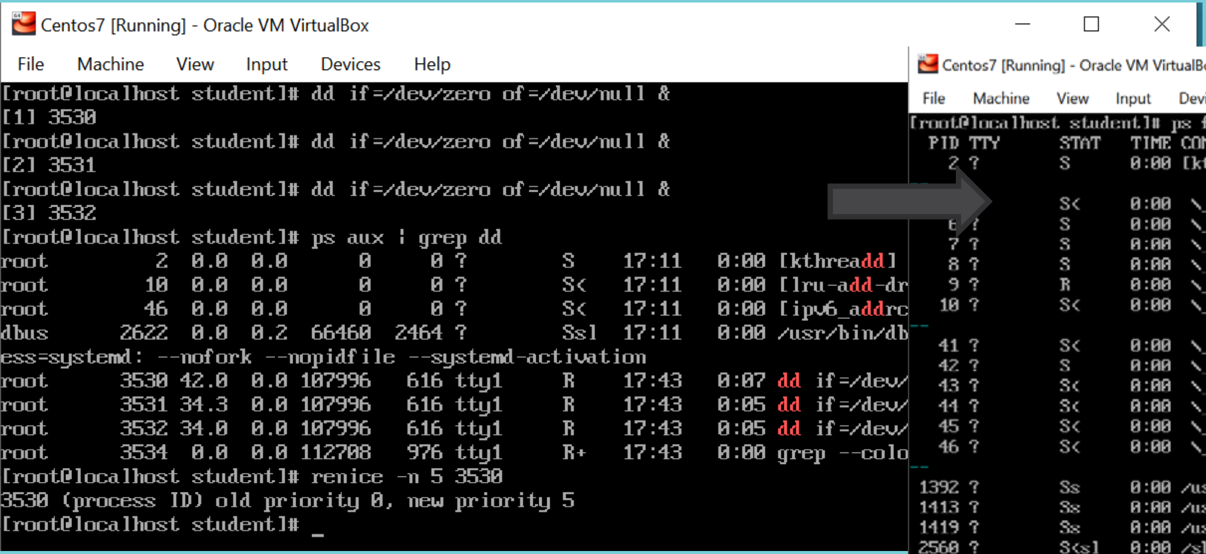
In a multithreaded environment, the urge to use ***killall*** is smaller. Because there is often just one process that is generating several threads, all these threads are terminated anyway by stopping the process that started them. You still can use ***killall*** , though, to terminate lots of processes with the same name that have been started on your server.

Process Management Basics

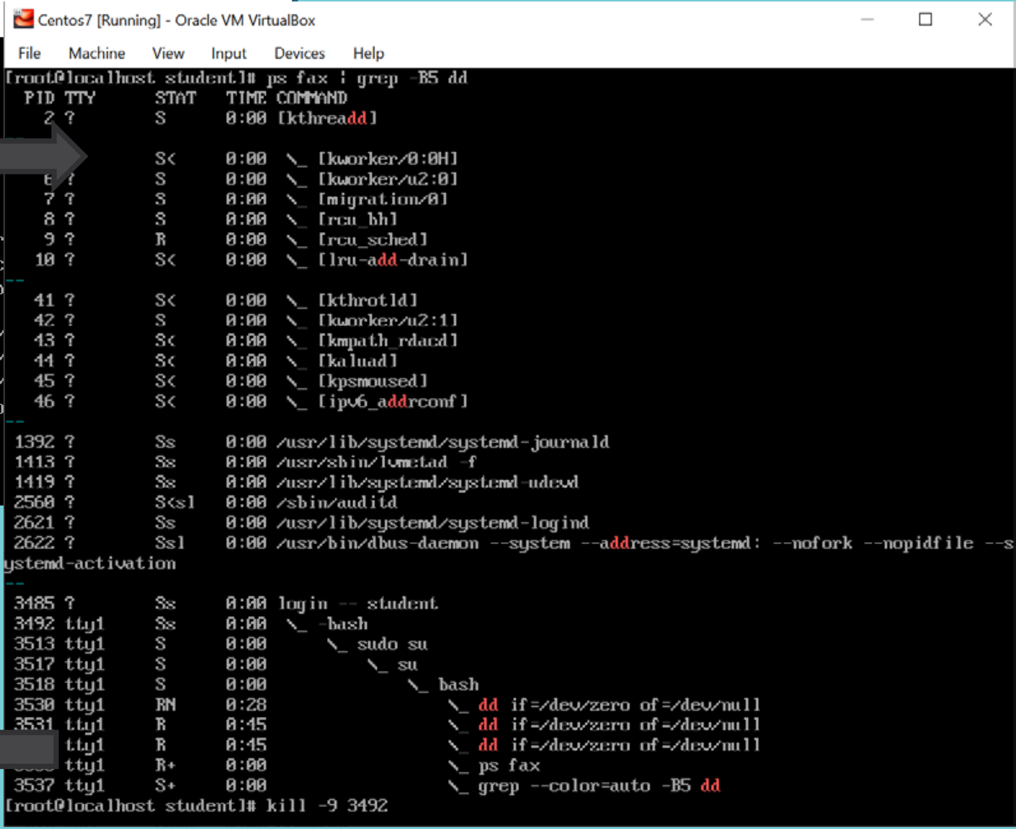
Example. Managing Processes from the Command Line
(how to work with ps, nice, kill, and related utilities to manage processes)

1. Open a root shell. From this shell, type ***dd if=/dev/zero of=/dev/null &*** . Repeat this command three times.
2. Type ***ps aux | grep dd*** . This shows all lines of output that have the letters ***dd*** in them; you will see more than just the ***dd*** processes, but that should not really matter. The processes you just started are listed last.
3. Use the PID of one of the ***dd*** processes to adjust the niceness, using ***renice -n 5*** <PID> . Notice that in top you cannot easily get an overview of processes and their current priority.
4. Type ***ps fax | grep -B5 dd*** . The ***-B5*** option shows the matching lines, including the five lines before that. Because ***ps fax*** shows hierarchical relationships between processes, you should also find the shell and its PID from which all the ***dd*** processes were started.
5. Find the PID of the shell from which the dd processes were started and type ***kill -9*** <PID> , replacing <PID> with the PID of the shell you just found. You will see that your root shell is closed, and with it, all of the ***dd*** processes. Killing a parent process is an easy and convenient way to kill all of its child processes also.

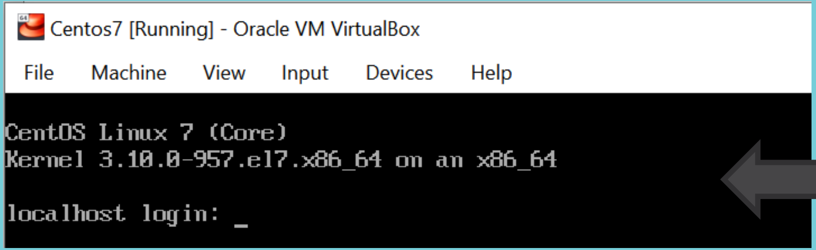
Process Management Basics



```
[root@localhost student]# dd if=/dev/zero of=/dev/null &
[1] 3530
[root@localhost student]# dd if=/dev/zero of=/dev/null &
[2] 3531
[root@localhost student]# dd if=/dev/zero of=/dev/null &
[3] 3532
[root@localhost student]# ps aux | grep dd
root         2  0.0  0.0      0     0 ?        S    17:11   0:00 [kthreadd]
root        10  0.0  0.0      0     0 ?        S<   17:11   0:00 [lru-add-dr
root        46  0.0  0.0      0     0 ?        S<   17:11   0:00 [ipv6_addrc
dbus      2622  0.0  0.2  66460  2464 ?        Ssl  17:11   0:00 /usr/bin/db
ess=systemd: --nofork --nopidfile --systemd-activation
root      3530 42.0  0.0 107996   616 tty1     R    17:43   0:07 dd if=/dev/
root      3531 34.3  0.0 107996   616 tty1     R    17:43   0:05 dd if=/dev/
root      3532 34.0  0.0 107996   616 tty1     R    17:43   0:05 dd if=/dev/
root      3534  0.0  0.0 112708   976 tty1     R+   17:43   0:00 grep --colo
[root@localhost student]# renice -n 5 3530
3530 (process ID) old priority 0, new priority 5
[root@localhost student]# _
```



```
[root@localhost student]# ps fax | grep -B5 dd
  PID TTY      STAT   TIME COMMAND
    2 ?        S      0:00 [kthreadd]
--
               S<     0:00  \_ [kworker/0:0H]
    6 ?        S      0:00  \_ [kworker/u2:0]
    7 ?        S      0:00  \_ [migration/0]
    8 ?        S      0:00  \_ [rcu_bh]
    9 ?        R      0:00  \_ [rcu_sched]
   10 ?        S<     0:00  \_ [lru-add-drain]
--
   41 ?        S<     0:00  \_ [kthrotld]
   42 ?        S      0:00  \_ [kworker/u2:1]
   43 ?        S<     0:00  \_ [kmpath_rdacd]
   44 ?        S<     0:00  \_ [kaluad]
   45 ?        S<     0:00  \_ [kpsmoused]
   46 ?        S<     0:00  \_ [ipv6_addrconf]
--
 1392 ?        Ss     0:00 /usr/lib/systemd/systemd-journald
 1413 ?        Ss     0:00 /usr/sbin/lvmetad -f
 1419 ?        Ss     0:00 /usr/lib/systemd/systemd-udevd
 2560 ?        S<sl   0:00 /sbin/auditd
 2621 ?        Ss     0:00 /usr/lib/systemd/systemd-logind
 2622 ?        Ssl    0:00 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --s
ystemd-activation
--
 3485 ?        Ss     0:00 login -- student
 3492 tty1     Ss     0:00  \_ -bash
 3513 tty1     S      0:00      \_ sudo su
 3517 tty1     S      0:00          \_ su
 3518 tty1     S      0:00              \_ bash
 3530 tty1     RN     0:28                  \_ dd if=/dev/zero of=/dev/null
 3531 tty1     R      0:45                  \_ dd if=/dev/zero of=/dev/null
      tty1     R      0:45                  \_ dd if=/dev/zero of=/dev/null
      tty1     R+     0:00                  \_ ps fax
 3537 tty1     S+     0:00                  \_ grep --color=auto -B5 dd
[root@localhost student]# kill -9 3492
```



```
CentOS Linux 7 (Core)
Kernel 3.10.0-957.el7.x86_64 on an x86_64

localhost login: _
```

Process Management Basics

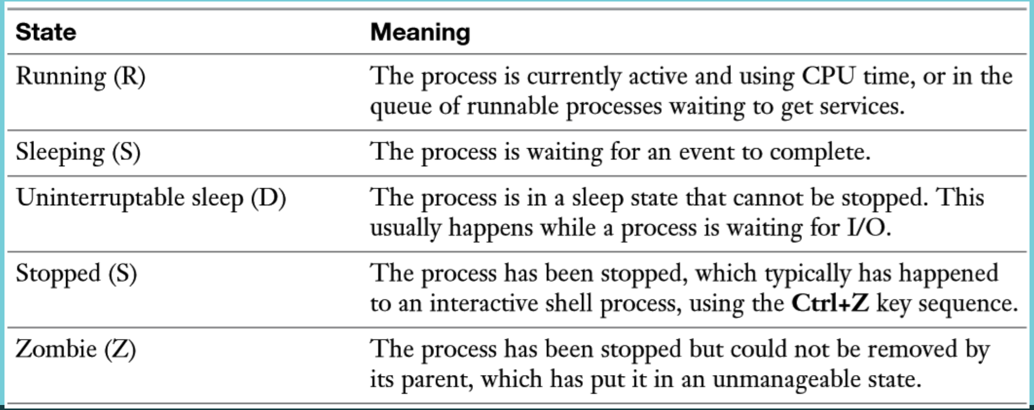
Using ***top*** to Manage Processes

A convenient tool to manage processes is ***top***.

For common process management tasks, top is so great because it gives an overview of the most active processes currently running (hence the name ***top***). This enables you to easily find processes that might need attention. From ***top***, you can also perform common process management tasks, such as adjusting the current process priority and killing processes.

Among the information that you can conveniently obtain from the top utility is the process state.

Followed tabled provides an overview of the different process states that you may be observing



State	Meaning
Running (R)	The process is currently active and using CPU time, or in the queue of runnable processes waiting to get services.
Sleeping (S)	The process is waiting for an event to complete.
Uninterruptable sleep (D)	The process is in a sleep state that cannot be stopped. This usually happens while a process is waiting for I/O.
Stopped (S)	The process has been stopped, which typically has happened to an interactive shell process, using the **Ctrl+Z** key sequence.
Zombie (Z)	The process has been stopped but could not be removed by its parent, which has put it in an unmanageable state.

Process Management Basics

Now that you know how to use the ***kill*** and ***nice*** commands from the command line, using the same functionality from top is even easier. From ***top***, type ***k*** . ***top*** will then prompt for the PID of the process you want to send a signal to. By default, the most active process is selected. After you enter the PID, top asks which signal you want to send. By default, signal 15 for SIGTERM is used.

However, if you want to insist a bit more, you can type 9 for SIGKILL. Then press Enter to terminate the process.

To renice a running process from top, type ***r*** . You are first prompted for the PID of the process you want to renice. After entering the PID, you are prompted for the nice value you want to use. Enter a positive value to increase process priority or a negative value to decrease process priority

Process Management Basics

For example:

<Shift>+<N> — sort by PID;

<Shift>+<P> — sort by CPU usage;

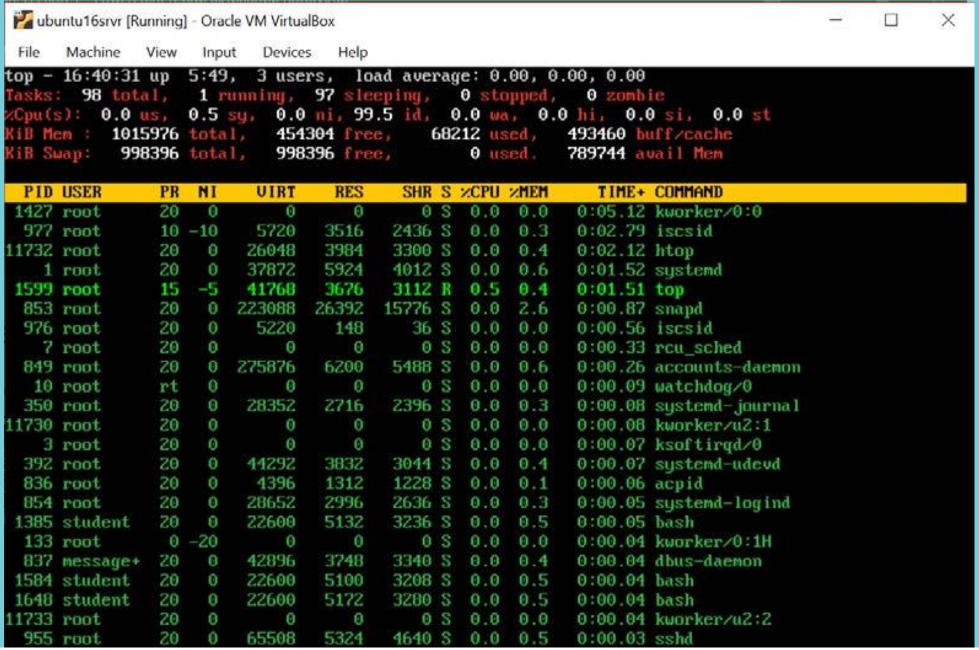
<Shift>+<M> — sort by Memory usage;

<Shift>+<T> — sort by Time usage;

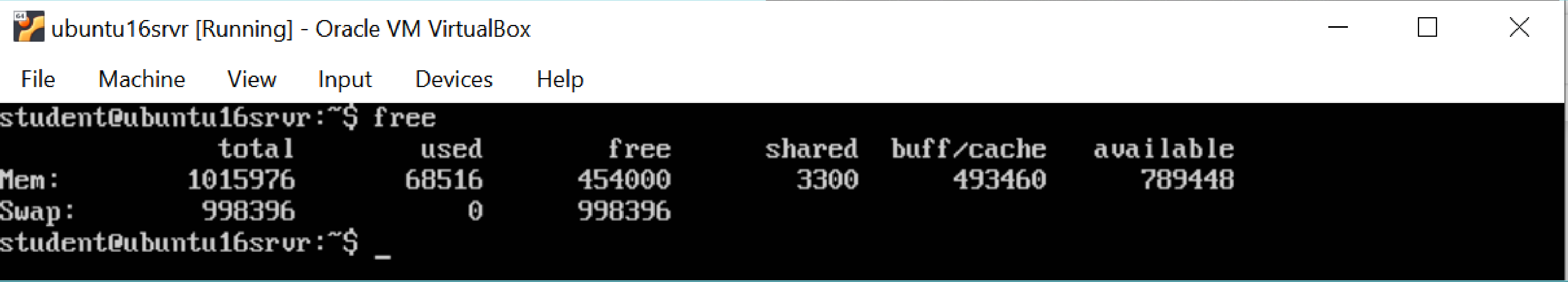
<Shift>+<Z> — change colors;

<c> - display absolute path of command;

And more hotkeys are available


ubuntu16srvr [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

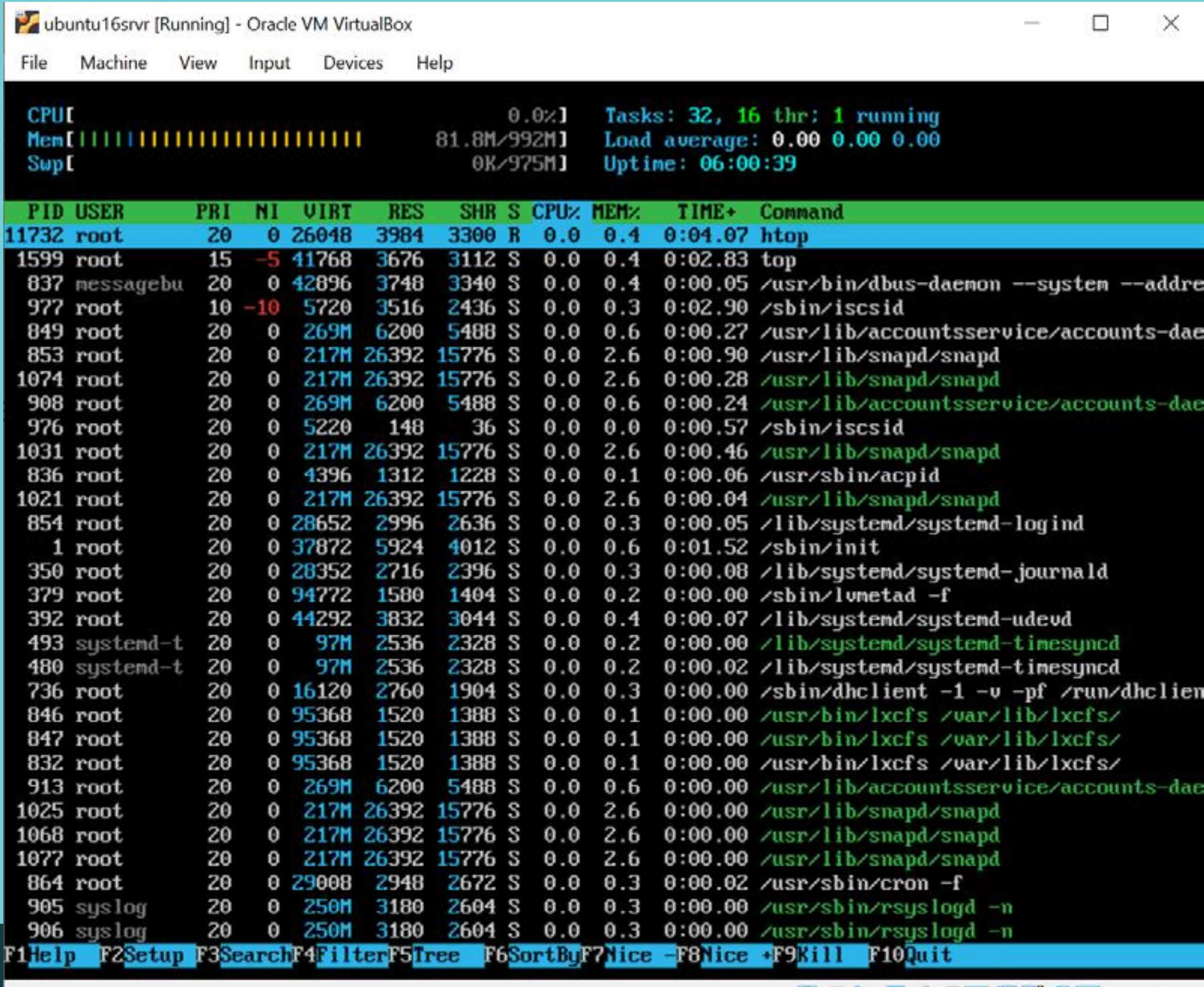
```
top - 16:40:31 up  5:49,  3 users,  load average: 0.00, 0.00, 0.00
Tasks:  98 total,   1 running,  97 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.5 sy,  0.0 ni, 99.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  1015976 total,   454304 free,    68212 used,   493460 buff/cache
KiB Swap:   998396 total,   998396 free,        0 used.   789744 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 1427 root      20   0       0      0      0 S  0.0  0.0   0:05.12 kworker/0:0
  977 root      10 -10    5720   3516   2436 S  0.0  0.3   0:02.79 iscsid
11732 root      20   0   26048   3984   3300 S  0.0  0.4   0:02.12 htop
    1 root      20   0   37872   5924   4012 S  0.0  0.6   0:01.52 systemd
 1599 root      15  -5   41768   3676   3112 R  0.5  0.4   0:01.51 top
  853 root      20   0  223088  26392  15776 S  0.0  2.6   0:00.87 snapd
  976 root      20   0    5220    148     36 S  0.0  0.0   0:00.56 iscsid
    7 root      20   0       0      0      0 S  0.0  0.0   0:00.33 rcu_sched
  849 root      20   0  275876   6200   5488 S  0.0  0.6   0:00.26 accounts-daemon
   10 root      rt   0       0      0      0 S  0.0  0.0   0:00.09 watchdog/0
  350 root      20   0   28352   2716   2396 S  0.0  0.3   0:00.08 systemd-journal
11730 root      20   0       0      0      0 S  0.0  0.0   0:00.08 kworker/u2:1
    3 root      20   0       0      0      0 S  0.0  0.0   0:00.07 ksoftirqd/0
  392 root      20   0   44292   3832   3044 S  0.0  0.4   0:00.07 systemd-udevd
  836 root      20   0    4396   1312   1228 S  0.0  0.1   0:00.06 acpid
  854 root      20   0   28652   2996   2636 S  0.0  0.3   0:00.05 systemd-logind
 1385 student   20   0   22600   5132   3236 S  0.0  0.5   0:00.05 bash
  133 root       0 -20       0      0      0 S  0.0  0.0   0:00.04 kworker/0:1H
  837 message+  20   0   42896   3748   3340 S  0.0  0.4   0:00.04 dbus-daemon
 1584 student   20   0   22600   5100   3208 S  0.0  0.5   0:00.04 bash
 1648 student   20   0   22600   5172   3280 S  0.0  0.5   0:00.04 bash
11733 root      20   0       0      0      0 S  0.0  0.0   0:00.04 kworker/u2:2
  955 root      20   0   65508   5324   4640 S  0.0  0.5   0:00.03 sshd
```


ubuntu16srvr [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

```
student@ubuntu16srvr:~$ free
              total        used        free      shared  buff/cache   available
Mem:        1015976       68516      454000        3300      493460      789448
Swap:        998396           0      998396
student@ubuntu16srvr:~$ _
```

Process Management Basics

Also frequently used: ***htop***



```
  CPU[                                     0.0%]   Tasks: 32, 16 thr: 1 running
  Mem[|||||||||||||||||||||||||||||   81.8M/992M]   Load average: 0.00 0.00 0.00
  Swp[                                   0K/975M]   Uptime: 06:00:39

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
11732 root       20   0 26048  3984  3300 R  0.0  0.4  0:04.07 htop
 1599 root       15  -5 41768  3676  3112 S  0.0  0.4  0:02.83 top
  837 messagebu  20   0 42896  3748  3340 S  0.0  0.4  0:00.05 /usr/bin/dbus-daemon --system --addre
  977 root       10 -10  5720  3516  2436 S  0.0  0.3  0:02.90 /sbin/iscsid
  849 root       20   0  269M  6200  5488 S  0.0  0.6  0:00.27 /usr/lib/accountsservice/accounts-dae
  853 root       20   0  217M 26392 15776 S  0.0  2.6  0:00.90 /usr/lib/snapd/snapd
 1074 root       20   0  217M 26392 15776 S  0.0  2.6  0:00.28 /usr/lib/snapd/snapd
  908 root       20   0  269M  6200  5488 S  0.0  0.6  0:00.24 /usr/lib/accountsservice/accounts-dae
  976 root       20   0  5220   148    36 S  0.0  0.0  0:00.57 /sbin/iscsid
 1031 root       20   0  217M 26392 15776 S  0.0  2.6  0:00.46 /usr/lib/snapd/snapd
  836 root       20   0  4396  1312  1228 S  0.0  0.1  0:00.06 /usr/sbin/acpid
 1021 root       20   0  217M 26392 15776 S  0.0  2.6  0:00.04 /usr/lib/snapd/snapd
  854 root       20   0 28652  2996  2636 S  0.0  0.3  0:00.05 /lib/systemd/systemd-logind
    1 root       20   0 37872  5924  4012 S  0.0  0.6  0:01.52 /sbin/init
  350 root       20   0 28352  2716  2396 S  0.0  0.3  0:00.08 /lib/systemd/systemd-journald
  379 root       20   0 94772  1580  1404 S  0.0  0.2  0:00.00 /sbin/lvmetad -f
  392 root       20   0 44292  3832  3044 S  0.0  0.4  0:00.07 /lib/systemd/systemd-udevd
  493 systemd-t  20   0   97M  2536  2328 S  0.0  0.2  0:00.00 /lib/systemd/systemd-timesyncd
  480 systemd-t  20   0   97M  2536  2328 S  0.0  0.2  0:00.02 /lib/systemd/systemd-timesyncd
  736 root       20   0 16120  2760  1904 S  0.0  0.3  0:00.00 /sbin/dhclient -1 -v -pf /run/dhclien
  846 root       20   0 95368  1520  1388 S  0.0  0.1  0:00.00 /usr/bin/lxcfs /var/lib/lxcfs/
  847 root       20   0 95368  1520  1388 S  0.0  0.1  0:00.00 /usr/bin/lxcfs /var/lib/lxcfs/
  832 root       20   0 95368  1520  1388 S  0.0  0.1  0:00.00 /usr/bin/lxcfs /var/lib/lxcfs/
  913 root       20   0  269M  6200  5488 S  0.0  0.6  0:00.00 /usr/lib/accountsservice/accounts-dae
 1025 root       20   0  217M 26392 15776 S  0.0  2.6  0:00.00 /usr/lib/snapd/snapd
 1068 root       20   0  217M 26392 15776 S  0.0  2.6  0:00.00 /usr/lib/snapd/snapd
 1077 root       20   0  217M 26392 15776 S  0.0  2.6  0:00.00 /usr/lib/snapd/snapd
  864 root       20   0 29008  2948  2672 S  0.0  0.3  0:00.02 /usr/sbin/cron -f
  905 syslog     20   0  250M  3180  2604 S  0.0  0.3  0:00.00 /usr/sbin/rsyslogd -n
  906 syslog     20   0  250M  3180  2604 S  0.0  0.3  0:00.00 /usr/sbin/rsyslogd -n
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

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.

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.

And and Or

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, in terminal but before that I need to check if the host is live or not.
ping -c3 www.tecmint.com && links www.tecmint.com

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.
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.
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.

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.

```
ls -l | less
```

```
echo -e "apple\npear\nbanana"|sort
```

QUESTIONS & ANSWERS

THANK YOU!