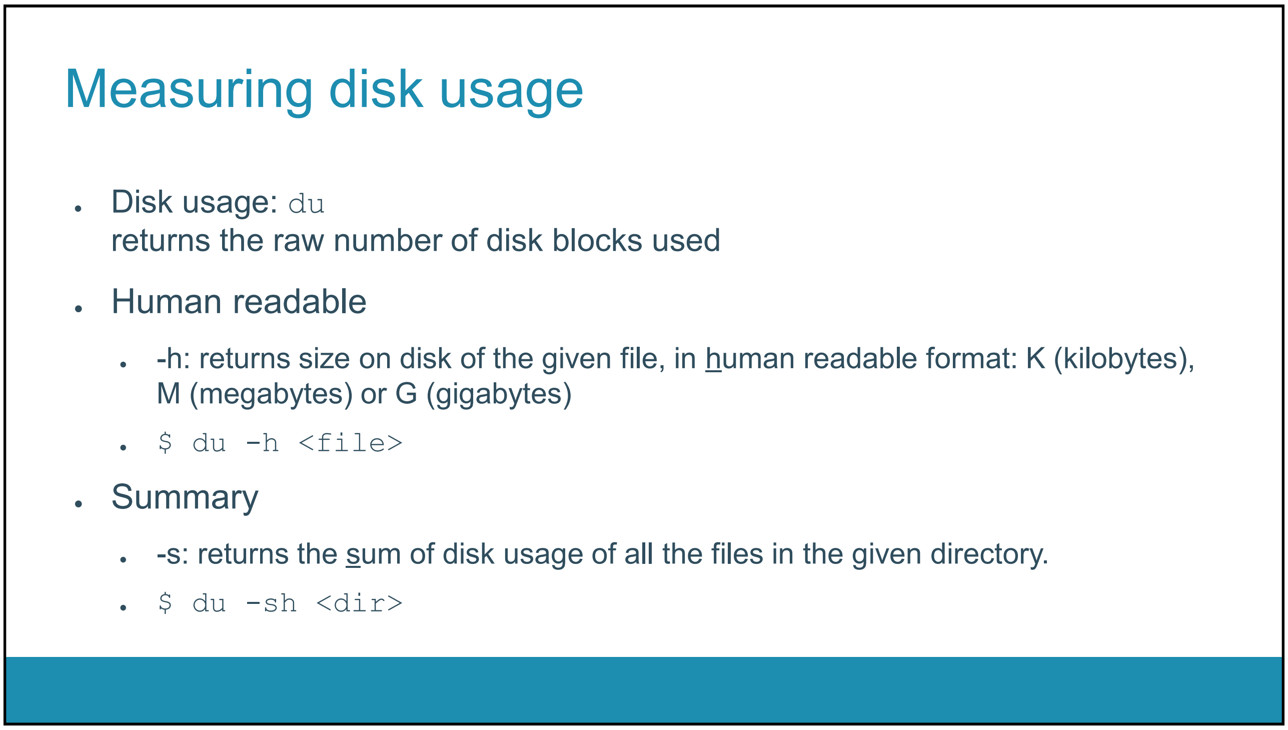
Outline

- Introduction - history
- Command line basics – getting help
- File system
- Working with files and directories
- More file handling
- The shell revisited
- ➢Monitoring resources

1

Check space

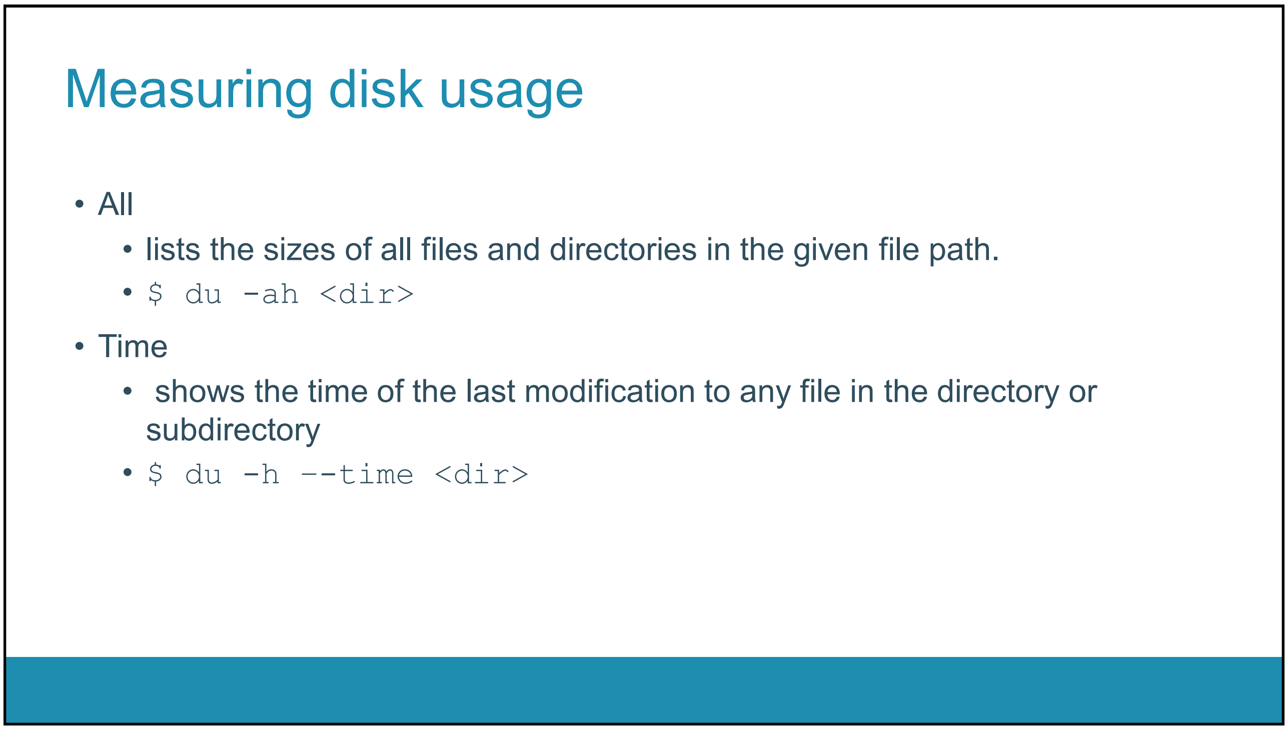
2



Measuring disk usage

- Disk usage: `du`
  returns the raw number of disk blocks used
- Human readable
  - -h: returns size on disk of the given file, in human readable format: K (kilobytes), M (megabytes) or G (gigabytes)
  - `$ du -h <file>`
- Summary
  - -s: returns the sum of disk usage of all the files in the given directory.
  - `$ du -sh <dir>`

3



Measuring disk usage

- All
  - lists the sizes of all files and directories in the given file path.
  - `$ du -ah <dir>`
- Time
  - shows the time of the last modification to any file in the directory or subdirectory
  - `$ du -h --time <dir>`

4

Measuring disk space

- Disk filesystem (Disk free): `df`
  full summary of available and used disk space usage of the file system on the Linux system.
- `$ df -h <dir>`
  Returns disk usage and free space for the filesystem containing the given directory.
- `$ df -h`
  Returns disk space information for all filesystems available in the system. When errors happen, useful to look for full filesystems.

5

Disk usage commands summary

- `du`:
  - provides detailed information about specific directories and files.
  - display the disk space used by files and directories within a specific directory. It can be used to get a summary of the disk space used by a particular directory and all of its subdirectories.
- `df`:
  - gives a high-level overview of disk usage across file systems.
  - display the amount of disk space available on a file system, as well as the total size and the amount of space used. df does not give information on specific files or directories, but rather on the file system as a whole.

6

List of open files: `lsof`

- `lsof`: shows open files and processes accessing them. Every object in Linux (e.g., devices, directories) is treated as a file.

Option	What	example
-u	Lists files opened by a specific user.	`lsof -u username`
-c	Lists files opened by a specific command.	`lsof -c command`
-p	Lists files opened by a specific process ID.	`lsof -p PID`

7

Memory

- `free`: check for memory RAM
- Result
  - **total**: total amount of physical RAM on your system.
  - **used**: amount of memory currently being utilized by running programs and processes.
  - **free**: amount of physical memory that is not currently being used by any running processes and is ready to be allocated to new processes.
  - **shared**: amount of memory used by the temporary "tmpfs" file system. tmpfs is a file system that stores files in the computer's main memory (RAM) making it faster to access compared to traditional storage methods like a hard drive.
  - **buff/cache**: memory that the kernel (operating system) uses to store recently used data so that it can be accessed quickly.
  - **available**: an estimate of how much memory is available for starting new applications, including memory used for buffers and caches that can be reclaimed.
- https://www.turing.com/kb/how-to-use-the-linux-free-command

8

Process management

11

jobs

- A **process** is an instance of a running program, managed by the Linux kernel.
  A **job** is a process or group of processes, managed by the shell.
  The shell provides commands like `jobs`, `fg` and `bg` to interact with jobs. The `ps` command is used to interact with processes.
- Control how your processes run with jobs
- A running program launched from the shell is known as a **job**.
  - is started from the command line
  - runs until the program completes its task.
- Each job is always in one of three states:
  - Foreground: Running, with control of the terminal. (default)
  - Background: Running, but not able to read from the terminal.
  - Stopped: Waiting to be resumed.

12

jobs

- Run a command, if you need to free up the terminal, you can stop the process. Ctrl-z stops a process

```
sleep 10000
^Z
```

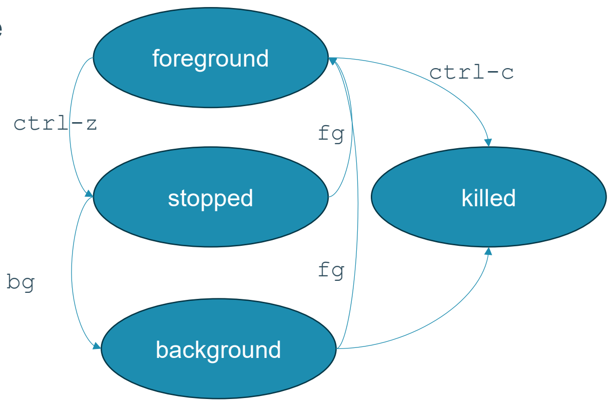
- Get a list of jobs in the background: `jobs`

```
jobs -l
```

Option `-l` shows info on the process id

- Bring a job back to the foreground: `fg` multiple stopped jobs, specify the job ID

```
fg 2
```



https://www.baeldung.com/linux/foreground-background-process#:~:text=A%20process%20that%20connects%20to,is%20called%20a%20background%20job.

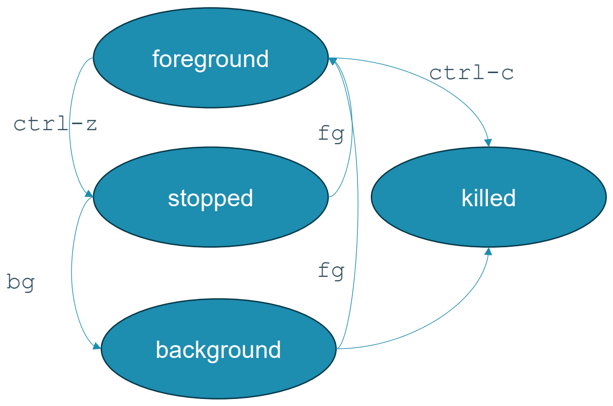
15

jobs

- Run a stopped command in the background
- Get a list of jobs in the background: `jobs`

```
jobs -l
```

- Bring a job to the background: `bg` multiple stopped jobs, specify the job ID
- `bg 2`
- Check with `jobs`: the process is in the background, but running instead of being stopped.



https://www.baeldung.com/linux/foreground-background-process#:~:text=A%20process%20that%20connects%20to,is%20called%20a%20background%20job.

16

&

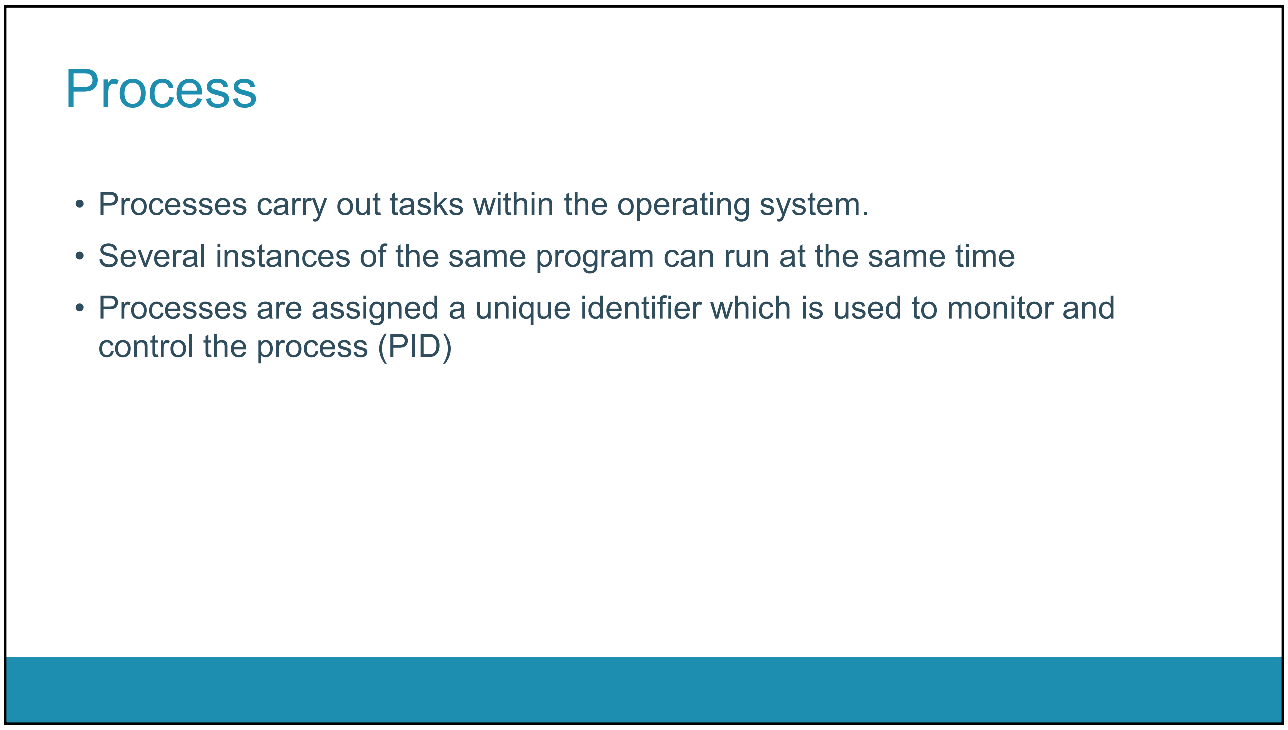
- **&** is a command line operator that instructs the shell to start the specified program in the background.
  - This allows you to have more than one program running at the same time without having to start multiple terminal sessions.
  - Starting a process in background: add & at the end of your line:
    ```
    $ sleep 10000 &
    ```

18

commands

- There are several commands that are used to control processes:
  - `jobs` - an alternate way of listing your own processes in background or suspended
  - `bg` - put a process in the background
  - `fg` - put a process in the foreground
  - `kill` - send a signal to one or more processes (usually to "kill" a process)
  - `ps` - list the processes running on the system

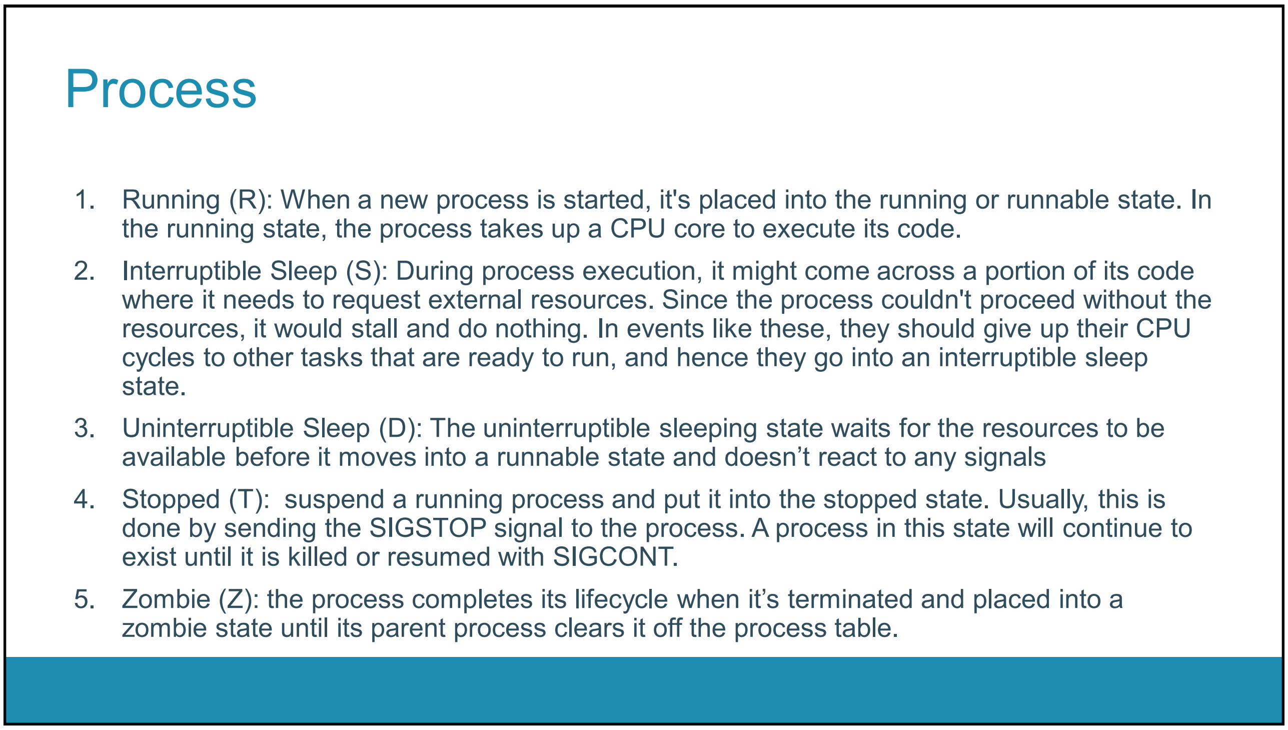
19



Process

- Processes carry out tasks within the operating system.
- Several instances of the same program can run at the same time
- Processes are assigned a unique identifier which is used to monitor and control the process (PID)

20



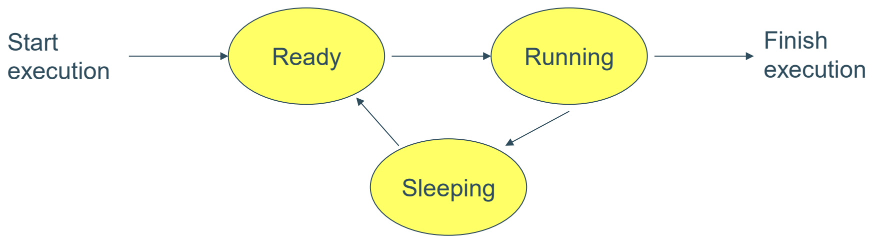
Process

1. Running (R): When a new process is started, it's placed into the running or runnable state. In the running state, the process takes up a CPU core to execute its code.
2. Interruptible Sleep (S): During process execution, it might come across a portion of its code where it needs to request external resources. Since the process couldn't proceed without the resources, it would stall and do nothing. In events like these, they should give up their CPU cycles to other tasks that are ready to run, and hence they go into an interruptible sleep state.
3. Uninterruptible Sleep (D): The uninterruptible sleeping state waits for the resources to be available before it moves into a runnable state and doesn't react to any signals
4. Stopped (T): suspend a running process and put it into the stopped state. Usually, this is done by sending the SIGSTOP signal to the process. A process in this state will continue to exist until it is killed or resumed with SIGCONT.
5. Zombie (Z): the process completes its lifecycle when it's terminated and placed into a zombie state until its parent process clears it off the process table.

21

Process

- A process in Linux is a running instance of a program, identified by a unique PID
- A Linux process can be in one of the following states:



ps

- **p**rocess **s**tatus: display running processes (snapshot)
  (cfr. Windows Task Manager ctrl-shift-esc)
- `$ps` Display the current user's processes
- `$ps -e` Display all processes running on the system
- `$ps -ef` Display detailed information about running processes
- `$ps -u [USER]` Display processes owned by the specified user
- `$ps -aux`

  a = show processes for all users
  u = display the process's user/owner
  x = also show processes not attached to a terminal

ps

```
  PID  TTY    STAT  TIME COMMAND
14748  pts/1  S     0:00 -bash
14795  pts/0  S     0:00 -bash
14974  pts/0  S     0:00 vi test1.txt
14876  pts/1  R     0:00 ps …
```

Process ID

Controlling Terminal name

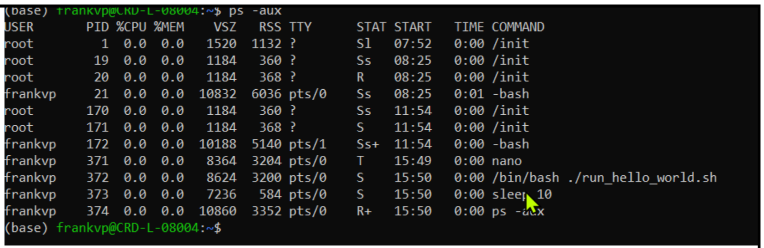
State:
S – Sleeping (waiting for input)
R – Running

Total CPU usage

Name of executable/command

- For the example above, both bash processes, which are the shell of both terminals, are waiting for the input of user. They must be in the sleeping state
- The vi process, which is an editor, is also waiting for the input of user. Hence it is also in sleeping state
- When ps reporting the processes in the system, it is the only process that is running. Hence it is in running state

ps -aux



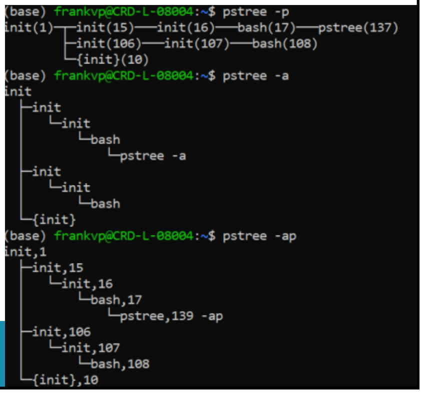
```
(base) frankvp@CRD-L-08004:~$ ps -aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   1520  1132 ?        Sl   07:52   0:00 /init
root        19  0.0  0.0   1184   360 ?        Ss   08:25   0:00 /init
root        20  0.0  0.0   1184   368 ?        R    08:25   0:00 /init
frankvp     21  0.0  0.0  10832  6036 pts/0    Ss   08:25   0:01 -bash
root       170  0.0  0.0   1184   360 ?        Ss   11:54   0:00 /init
root       171  0.0  0.0   1184   368 ?        S    11:54   0:00 /init
frankvp    172  0.0  0.0  10188  5140 pts/1    Ss+  11:54   0:00 -bash
frankvp    371  0.0  0.0   8364  3204 pts/0    T    15:49   0:00 nano
frankvp    372  0.0  0.0   8624  3200 pts/0    S    15:50   0:00 /bin/bash ./run_hello_world.sh
frankvp    373  0.0  0.0   7236   584 pts/0    S    15:50   0:00 sleep 10
frankvp    374  0.0  0.0  10860  3352 pts/0    R+   15:50   0:00 ps -aux
(base) frankvp@CRD-L-08004:~$
```

- More fields:

- USER: The effective user
- PID: Process ID
- %CPU: CPU time used divided by the time the process has been running
- %MEM: Ratio of the process's resident set size to the physical memory on the machine
- VSZ: Virtual memory usage of the entire process
- RSS: Resident set size, the non-swapped physical memory that a task has used
- TTY: Controlling terminal associated with the process
- STAT: Process status code
- START: Start time of the process
- TIME: Total CPU usage time
- COMMAND: Name of executable/command

pstree

- Shows the running processes as a tree, making the output more visually appealing.
- The root of the tree is either init or the process with the given pid.
- Options
  - -a: to include command line arguments in output
  - -p: display PIDs for each process name
  - -u: who is the owner/user of a process



```
(base) frankvp@CRD-L-08004:~$ pstree -p
init(1)─┬─init(15)───init(16)───bash(17)───pstree(137)
        ├─init(106)───init(107)───bash(108)
        └─{init}(10)
(base) frankvp@CRD-L-08004:~$ pstree -a
init
  ├─init
  │   └─init
  │       └─bash
  │           └─pstree -a
  ├─init
  │   └─init
  │       └─bash
  └─{init}
(base) frankvp@CRD-L-08004:~$ pstree -ap
init,1
  ├─init,15
  │   └─init,16
  │       └─bash,17
  │           └─pstree,139 -ap
  ├─init,106
  │   └─init,107
  │       └─bash,108
  └─{init},10
```

27

Signals

- Signals are used for communication between processes
- Use the `kill` command to send various signals to a running process.
- `kill -l`: get a list of all the available signals you can send to a process
- Some common use cases:
  - Interrupting a Process (SIGINT): to interrupt a process, typically via Ctrl+C. Often used to stop a command that's taking too long to complete or to stop a process that's running in the foreground.
  - Terminating a Process (SIGTERM): a generic signal used to terminate a program, can be ignored by the program.
  - Forcefully Terminating a Process (SIGKILL): forcefully terminates a process and cannot be ignored.
  - Pausing and Resuming a Process (SIGSTOP and SIGCONT): SIGSTOP pauses a process, and the SIGCONT signal resumes the execution of a paused process, to move processes between the foreground and background.

29

kill

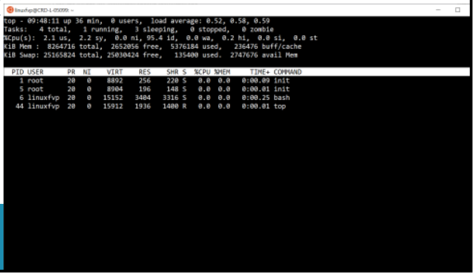
- Send (default) a terminate signal to the given processes.
  - `kill <pid>`
- Send an immediate termination signal. Useful when a process is really stuck.
  - `kill -9 <pid>`
- Suspend a process by sending STOP signal
  - `kill -STOP <PID> or %<jobID>`
  - `kill -19 <PID> or %<jobID>`
- Resume the process, use the CONT flag
  - `kill -CONT <PID> or %<jobID>`
  - `kill -18 <PID> or %<jobID>`

30

top

- Displays a real-time system status summary. The output displays the amount of system memory(RAM) used for different purposes, percentage of CPU being utilized, swap memory, and other information.
- Press ‘z‘ option will display the running process in color which may help you to identify the running process easily.
- Press ‘f’ to edit the columns, press space bar to select/deselect

http://www.thegeekstuff.com/2010/01/15-practical-unix-linux-top-command-examples/



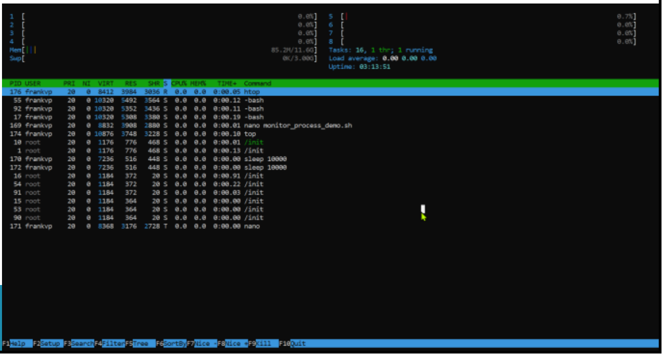
```
top - 09:48:11 up 36 min,  0 users,  load average: 0.52, 0.58, 0.59
Tasks:   4 total,   1 running,   3 sleeping,   0 stopped,   0 zombie
%Cpu(s):  2.1 us,  2.2 sy,  0.0 ni, 95.4 id,  0.0 wa,  0.2 hi,  0.0 si,  0.0 st
KiB Mem :  8264716 total,  2652056 free,  5376184 used,   236476 buff/cache
KiB Swap: 25165824 total, 25030424 free,   135400 used.  2747676 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0    8892    256    220 S   0.0  0.0   0:00.09 init
    5 root      20   0    8904    196    148 S   0.0  0.0   0:00.01 init
    6 linuxfvp  20   0   15152   3404   3316 S   0.0  0.0   0:00.25 bash
   44 linuxfvp  20   0   15912   1936   1400 R   0.0  0.0   0:00.01 top
```

31

htop

- Displays the data in a more informative and interactive manner.
- The process names are more descriptive and the mouse integration is an extra feature that is not present with the ‘top’ command.
- Use the mouse to select various columns displayed on the terminal output.



```
1  [                                              0.0%]   5  [|                                             0.7%]
2  [                                              0.0%]   6  [                                              0.0%]
3  [                                              0.0%]   7  [                                              0.0%]
4  [                                              0.0%]   8  [                                              0.0%]
Mem[|||                                    85.2M/11.6G]   Tasks: 16, 1 thr; 1 running
Swp[                                          0K/3.00G]   Load average: 0.00 0.00 0.00
                                                          Uptime: 03:13:51

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
  176 frankvp    20   0  8412  3984  3036 R  0.0  0.0  0:00.05 htop
   55 frankvp    20   0 10320  5492  3564 S  0.0  0.0  0:00.12 -bash
   92 frankvp    20   0 10320  5352  3436 S  0.0  0.0  0:00.11 -bash
   17 frankvp    20   0 10320  5308  3380 S  0.0  0.0  0:00.19 -bash
  169 frankvp    20   0  8832  3908  2880 S  0.0  0.0  0:00.01 nano monitor_process_demo.sh
  174 frankvp    20   0 10876  3748  3228 S  0.0  0.0  0:00.10 top
   10 root       20   0  1176   776   468 S  0.0  0.0  0:00.01 /init
    1 root       20   0  1176   776   468 S  0.0  0.0  0:00.13 /init
  170 frankvp    20   0  7236   516   448 S  0.0  0.0  0:00.00 sleep 10000
  172 frankvp    20   0  7236   516   448 S  0.0  0.0  0:00.00 sleep 10000
   16 root       20   0  1184   372    20 S  0.0  0.0  0:00.91 /init
   54 root       20   0  1184   372    20 S  0.0  0.0  0:00.22 /init
   91 root       20   0  1184   372    20 S  0.0  0.0  0:00.03 /init
   15 root       20   0  1184   364    20 S  0.0  0.0  0:00.00 /init
   53 root       20   0  1184   364    20 S  0.0  0.0  0:00.00 /init
   90 root       20   0  1184   364    20 S  0.0  0.0  0:00.00 /init
  171 frankvp    20   0  8368  3176  2728 T  0.0  0.0  0:00.00 nano

F1Help  F2Setup  F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```