Processes

Viewing Processes

ps commands is used

```
PID TTY          TIME CMD

7838 pts/0    00:00:00 bash

7960 pts/0    00:00:00 ps
```

and to see the processes of others user we need to use ps aux

```
ubuntu@ubuntu:~$ ps aux

USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND

root         1  0.0  0.8  34224  8164 ?        Ss   04:23   0:20 /sbin/init fixrtc splash splash

root         2  0.0  0.0      0     0 ?        S    04:23   0:00 [kthreadd]

root         3  0.0  0.0      0     0 ?        I<   04:23   0:00 [rcu_gp]

root         4  0.0  0.0      0     0 ?        I<   04:23   0:00 [rcu_par_gp]

root         8  0.0  0.0      0     0 ?        I<   04:23   0:00 [mm_percpu_wq]

root         9  0.0  0.0      0     0 ?        S    04:23   0:04 [ksoftirqd/0]

root        10  0.0  0.0      0     0 ?        I    04:23   0:06 [rcu_preempt]

root        11  0.0  0.0      0     0 ?        S    04:23   0:00 [migration/0]

root        12  0.0  0.0      0     0 ?        S    04:23   0:00 [idle_inject/0]

root        14  0.0  0.0      0     0 ?        S    04:23   0:00 [cpuhp/0]

root        15  0.0  0.0      0     0 ?        S    04:23   0:00 [cpuhp/1]

root        16  0.0  0.0      0     0 ?        S    04:23   0:00 [idle_inject/1]

root        17  0.0  0.0      0     0 ?        S    04:23   0:00 [migration/1]

root        18  0.0  0.0      0     0 ?        S    04:23   0:01 [ksoftirqd/1]

root        20  0.0  0.0      0     0 ?        I<   04:23   0:03 [kworker/1:0H-kblockd]
root         21  0.0  0.0      0     0 ?        S    04:23   0:00 [cpuhp/2]
root         22  0.0  0.0      0     0 ?        S    04:23   0:00 [idle_inject/2]
root         23  0.0  0.0      0     0 ?        S    04:23   0:00 [migration/2]
root         24  0.0  0.0      0     0 ?        S    04:23   0:02 [ksoftirqd/2]
root         27  0.0  0.0      0     0 ?        S    04:23   0:00 [cpuhp/3]
root         28  0.0  0.0      0     0 ?        S    04:23   0:00 [idle_inject/3]
root         29  0.0  0.0      0     0 ?        S    04:23   0:00 [migration/3]
```

other very useful command is the top command; top gives you real-time statistics about the processes running on your system instead of a one-time view. These statistics will refresh every 10 seconds, but will also refresh when you use the arrow keys to browse the various rows. Another great command to gain insight into your system is via the `top` command

```
Dvs:~ dvs$ top

Processes: 319 total, 2 running, 317 sleeping, 1289 threads         15:35:12
Load Avg: 2.43, 3.00, 2.89  CPU usage: 1.42% user, 1.66% sys, 96.90% idle
SharedLibs: 240M resident, 56M data, 67M linkedit.
MemRegions: 164107 total, 2709M resident, 173M private, 662M shared.
PhysMem: 7569M used (1138M wired), 622M unused.
VM: 13T vsize, 1090M framework vsize, 374(0) swapins, 374(0) swapouts.
Networks: packets: 228775/253M in, 63077/8410K out.
Disks: 157018/5355M read, 99849/1734M written.

PID    COMMAND      %CPU TIME     #TH    #WQ  #PORT MEM    PURG   CMPRS  PGRP
2465   top          3.4  00:03.18 1/1    0    23    2872K  0B     0B     2465
2464   Google Chrom 0.0  00:00.19 10     1    94    18M    4096B  0B     804
2463   Google Chrom 0.0  00:00.65 13     1    131   23M+   4096B  0B     804
2451   CoreServices 0.0  00:00.12 3      1    139   3088K  0B     0B     2451
2443   com.apple.sp 0.0  00:00.39 2      1    47    15M    0B     0B     2443
```

```
2432  mdworker     0.0  00:00.08 4     2    52    3148K  0B     0B     2432
2431  mdworker     0.0  00:00.08 3     1    51    3124K  0B     0B     2431
2424  Google Chrom 0.0  00:00.46 9     1    76    14M    4096B  0B     804
2421  com.apple.iC 0.0  00:00.50 2     1    55    3528K  0B     0B     2421
2401  ocspd        0.0  00:00.21 2     1    34    1692K  0B     0B     2401
2395  Google Chrom 0.1  14:56.93 15    1    287   277M+  28K    0B     804
2392  mdworker     0.0  00:00.11 3     1    54    3336K  0B     0B     2392
2378  mdworker     0.0  00:00.54 3     1    54    3540K  0B     0B     2378
2377  mdworker     0.0  00:00.53 3     1    54    3532K  0B     0B     2377
2375  netbiosd     0.0  00:00.06 2     2    26    2540K  0B     0B     2375
2360  bash         0.0  00:00.04 1     0    19    852K   0B     0B     2360
2359  login        0.0  00:00.02 2     1    30    1564K  0B     0B     2359
2328  mdworker     0.0  00:00.63 3     1    54    3616K  0B     0B     2328
1392  Magnet       0.0  00:04.09 4     2    176   8116K  0B     0B     1392
944   com.apple.au 0.0  00:00.02 2     2    16    888K   0B     0B     944
918   bash         0.0  00:00.01 1     0    19    840K   0B     0B     918
917   login        0.0  00:00.02 2     1    29    1564K  0B     0B     917
916   Terminal     1.7  00:31.25 8     3    331   28M    4652K  0B     916
913   Google Chrom 0.0  00:09.19 8     1    150   13M    4096B  0B     804
901   mdworker     0.0  00:00.69 3     1    54    3736K  0B     0B     901
897   mdworker     0.0  00:01.02 3     1    55    3604K  0B     0B     897
888   Google Chrom 0.1  00:53.16 14    1    200   104M   0B     0B     804
864   VTDecoderXPC 0.0  00:00.08 2     1    45    2780K  0B     0B     864
863   Obsidian Hel 0.0  00:59.82 25    1    212   83M    0B     0B     860
862   Obsidian Hel 0.0  00:00.32 8     1    87    11M    0B     0B     860
861   Obsidian Hel 0.0  00:19.38 10    1    139   48M    13M    0B
```

Managing Processes

You can send signals that terminate processes; there are a variety of types of signals that correlate to exactly how "cleanly" the process is dealt with by the kernel. To kill a command, we can use the appropriately named `kill` command and the associated PID that we wish to kill. i.e., to kill PID 1337, we'd use `kill 1337`.

```
kill 1337
```

Below are some of the signals that we can send to a process when it is killed:

- SIGTERM - Kill the process, but allow it to do some cleanup tasks beforehand
- SIGKILL - Kill the process - doesn't do any cleanup after the fact
- SIGSTOP - Stop/suspend a process

How do Processes Start?

Let's start off by talking about namespaces. The Operating System (OS) uses namespaces to ultimately split up the resources available on the computer to (such as CPU, RAM and priority) processes. Think of it as splitting your computer up into slices -- similar to a cake. Processes within that slice will have access to a certain amount of computing power, however, it will be a small portion of what is actually available to every process overall.

Namespaces are great for security as it is a way of isolating processes from another -- only those that are in the same namespace will be able to see each other.

We previously talked about how PID works, and this is where it comes into play. The process with an ID of 0 is a process that is started when the system boots. This process is the system's init on Ubuntu, such as **systemd**, which is used to provide a way of managing a user's processes and sits in between the operating system and the user.

For example, once a system boots and it initialises, **systemd** is one of the first processes that are started. Any program or piece of software that we want to start will start as what's known as a child process of **systemd**. This means that it is controlled by **systemd**, but will run as its own process (although sharing the resources from **systemd**) to make it easier for us to identify and the likes.

Getting Processes/Services to Start on Boot

Some applications can be started on the boot of the system that we own. For example, web servers, database servers or file transfer servers. This software is often critical and is often told to start during the boot-up of the system by administrators.

In this example, we're going to be telling the apache web server to be starting apache manually and then telling the system to launch apache2 on boot.

Enter the use of `systemctl` -- this command allows us to interact with the **systemd** process/daemon. Continuing on with our example, systemctl is an easy to use command that takes the following formatting: `systemctl [option] [service]`

For example, to tell apache to start up, we'll use `systemctl start apache2`. Seems simple enough, right? Same with if we wanted to stop apache, we'd just replace the `[option]` with stop (instead of start like we provided)

We can do four options with `systemctl`:

- Start
- Stop
- Enable
- Disable

An Introduction to Backgrounding and Foregrounding in Linux

Processes can run in two states: In the background and in the foreground. For example, commands that you run in your terminal such as "echo" or things of that sort will run in the foreground of your terminal as it is the only command provided that hasn't been told to run in the background. "Echo" is a great example as the output of echo will return to you in the foreground, but wouldn't in the background - take the screenshot below, for example.

- Forground Process

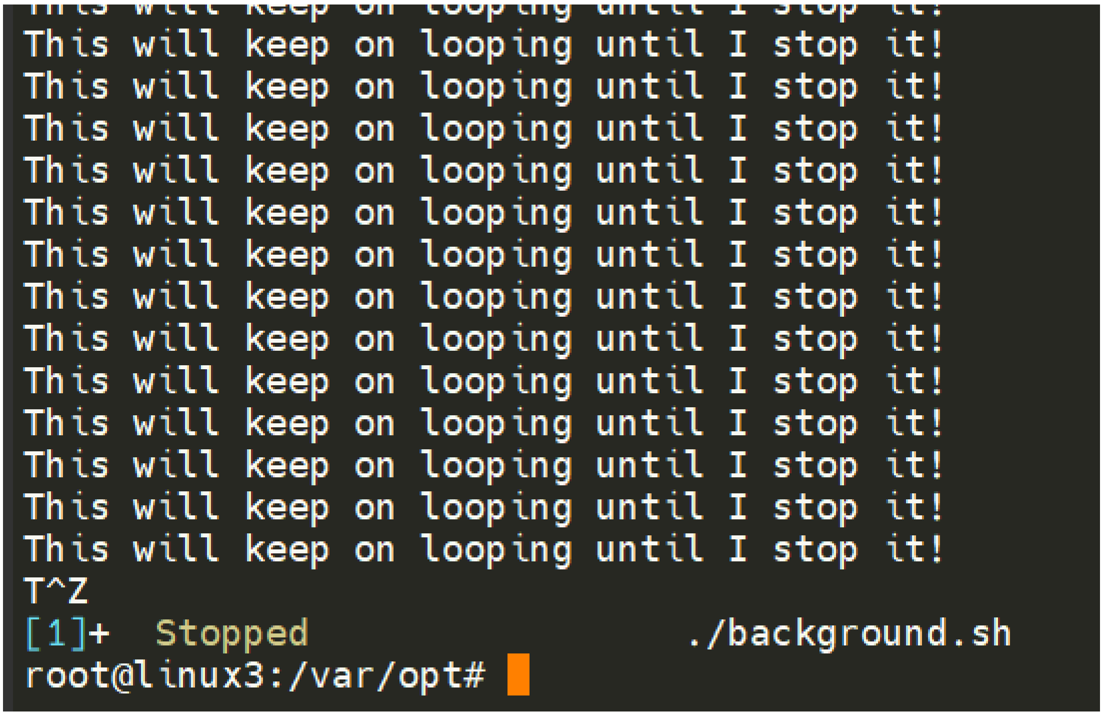
```
Dvs:~ dvs$ echo Divyesh

Divyesh
```

Here we're running `echo "Hi THM"` , where we expect the output to be returned to us like it is at the start. But after adding the `&` operator to the command, we're instead just given the ID of the echo process rather than the actual output -- as it is running in the background.

This is great for commands such as copying files because it means that we can run the command in the background and continue on with whatever further commands we wish to execute (without having to wait for the file copy to finish first)

We can do the exact same when executing things like scripts -- rather than relying on the & operator, we can use `Ctrl + Z` on our keyboard to background a process. It is also an effective way of "pausing" the execution of a script or command like in the example below:



```
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
This will keep on looping until I stop it!
T^Z
[1]+  Stopped                 ./background.sh
root@linux3:/var/opt#
```

This script will keep on repeating "This will keep on looping until I stop!" until I stop or suspend the process. By using `Ctrl + Z` (as indicated by **T^Z**). Now our terminal is no longer filled up with messages -- until we foreground it, which we will discuss below.