



Unidad 2

Procesos en Linux

Tipos de Procesos

Existen fundamentalmente 2 tipos de procesos en Linux:

- En primer plano - también llamados inactivos. estos procesos fueron inicializados y se encuentran controlados por una sesión de terminal.
- Segundo plano - también llamados no interactivos o automáticos. Son procesos que no están “conectados” con una sesión de terminal, no están esperando ninguna interacción por parte del usuario.

Que son los Daemons?

Son procesos especiales que se ejecutan en segundo plano pero que son arrancados al inicio del sistema y deben ser ejecutados siempre, “nunca mueren”. Están configurados para dar algún servicio en particular y pueden ser controlados por el usuario mediante el proceso de arranque del sistema (init).

Estados de los procesos en Linux?

Durante la ejecución de un proceso cambia de estado dependiendo del entorno y las circunstancias. En Linux, los procesos están en alguno de los siguientes estados:

- En Ejecución - en este caso puede estar ejecutándose, es decir que es el proceso que está en el procesador, o puede estar listo es decir que está a la espera de que el procesador se libere para ingresar.



- **En Espera** - en este estado, un proceso está inactivo porque necesita que algo suceda, puede estar esperando algún evento de un recurso de sistema o puede estar esperando alguna acción del usuario. Adicionalmente, el kernel también diferencia entre dos tipos de procesos en espera: interrumpible, es decir que se puede modificar mediante una señal (las veremos después) e in-interrumpible que está esperando alguna acción del hardware y no puede ser modificado mediante una señal.
- **Detenido** - en este estado el proceso ha finalizado, generalmente por haber recibido una señal para hacerlo.
- **Zombie** - estos son procesos que están muertos, que no consumen tiempo de procesador pero todavía están presentes en la tabla de procesos que mantiene el kernel.

Como identificar procesos en Linux

Dado que Linux es un sistema operativo multiusuario, cada instancia de proceso debe ser identificado de forma inequívoca por el Kernel.

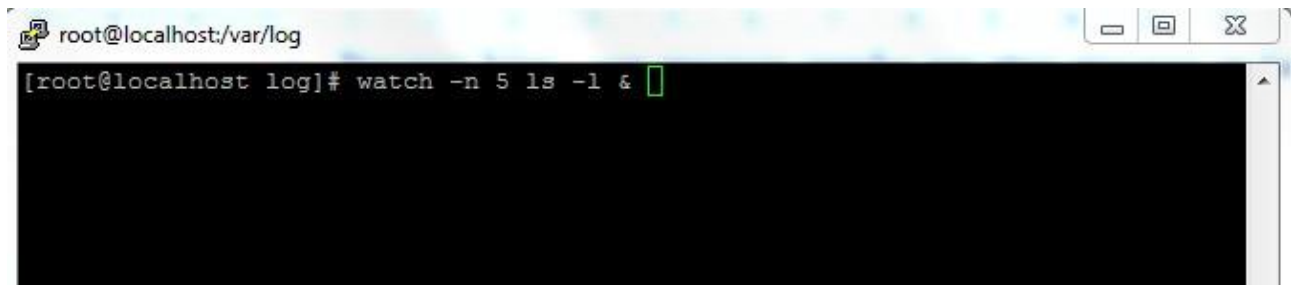
Aquí es donde cobra sentido el PID (Process id) que es un número que identifica a cada proceso como así también el PPID (parent process id, proceso padre) que sería el número del proceso padre, de aquí que los procesos también se pueden categorizar como:

- **Procesos Padre** - son procesos que crean subprocesos que dependen directamente de él.
- **Procesos hijos** - son procesos creados por otros procesos en tiempo de ejecución.



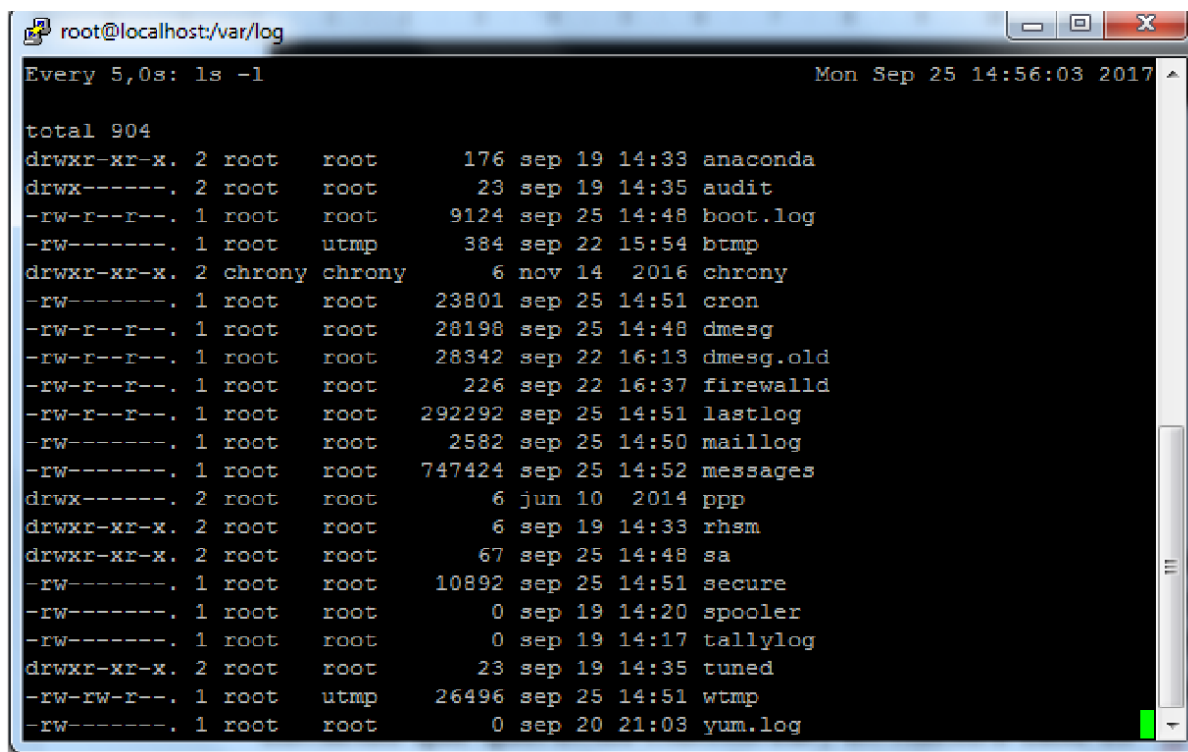
Administrar Procesos en Segundo Plano

Para comenzar un proceso en segundo plano, se debe adjuntar el símbolo “&” luego del comando que vayamos a ejecutar, el proceso que hayamos enviado a segundo plano no leerá ninguna entrada del usuario hasta que lo enviemos nuevamente a primer plano.



```
root@localhost:/var/log  
[root@localhost log]# watch -n 5 ls -l &
```

En el ejemplo de arriba pueden ver que utilizamos el programa “watch” que nos permite ejecutar un programa en un intervalo de tiempo determinado con el flag “-n” en este caso le hemos especificado cada 5 segundos. Como le adjuntamos el signo “&” al finalizar la sentencia del comando que queremos monitorear, en nuestro caso es “ls -l” lo enviaremos a segundo plano. Si no lo hiciéramos veríamos el siguiente resultado:

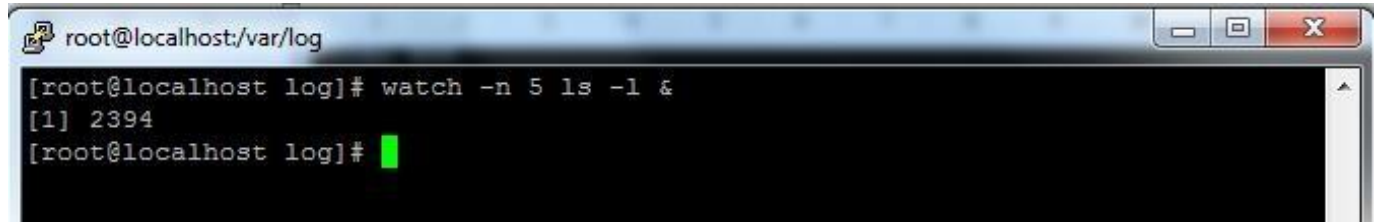


```
root@localhost:/var/log  
Every 5.0s: ls -l  
Mon Sep 25 14:56:03 2017  
total 904  
drwxr-xr-x. 2 root root 176 sep 19 14:33 anaconda  
drwx-----. 2 root root 23 sep 19 14:35 audit  
-rw-r--r--. 1 root root 9124 sep 25 14:48 boot.log  
-rw-----. 1 root utmp 384 sep 22 15:54 bttmp  
drwxr-xr-x. 2 chrony chrony 6 nov 14 2016 chrony  
-rw-----. 1 root root 23801 sep 25 14:51 cron  
-rw-r--r--. 1 root root 28198 sep 25 14:48 dmesg  
-rw-r--r--. 1 root root 28342 sep 22 16:13 dmesg.old  
-rw-r--r--. 1 root root 226 sep 22 16:37 firewallld  
-rw-r--r--. 1 root root 292292 sep 25 14:51 lastlog  
-rw-----. 1 root root 2582 sep 25 14:50 maillog  
-rw-----. 1 root root 747424 sep 25 14:52 messages  
drwx-----. 2 root root 6 jun 10 2014 ppp  
drwxr-xr-x. 2 root root 6 sep 19 14:33 rhsm  
drwxr-xr-x. 2 root root 67 sep 25 14:48 sa  
-rw-----. 1 root root 10892 sep 25 14:51 secure  
-rw-----. 1 root root 0 sep 19 14:20 spooler  
-rw-----. 1 root root 0 sep 19 14:17 tallylog  
drwxr-xr-x. 2 root root 23 sep 19 14:35 tuned  
-rw-rw-r--. 1 root utmp 26496 sep 25 14:51 wtmp  
-rw-----. 1 root root 0 sep 20 21:03 yum.log
```



Como era de esperarse, simplemente se trata de un “`ls -l`” que se ejecuta cada 5 segundos.

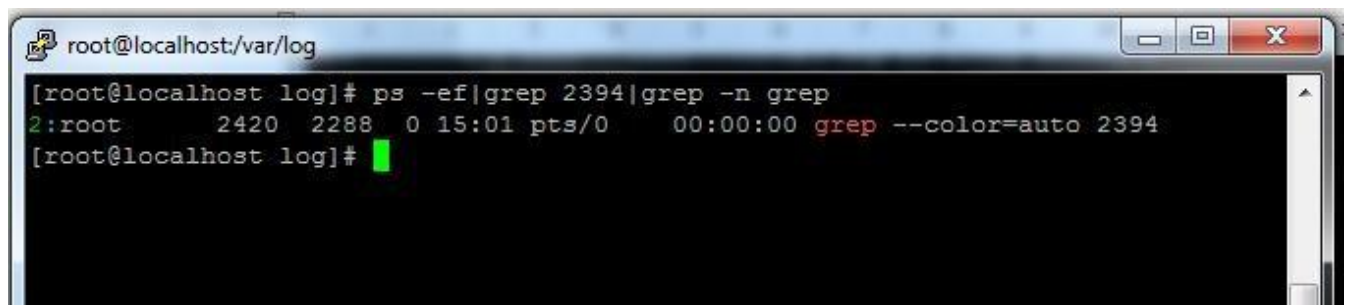
Al mandarlo a segundo plano veremos lo siguiente:



```
root@localhost:/var/log
[root@localhost log]# watch -n 5 ls -l &
[1] 2394
[root@localhost log]#
```

El “[1]” significa que hay un solo proceso en segundo plano iniciado por este usuario, y el número que le sigue es el PID del proceso

Al ejecutar un `ps -ef` y filtrar por el número de proceso lo podemos encontrar:



```
root@localhost:/var/log
[root@localhost log]# ps -ef | grep 2394 | grep -n grep
2:root      2420  2288  0 15:01 pts/0    00:00:00 grep --color=auto 2394
[root@localhost log]#
```

Luego utilizaremos los comandos “`bg`” y “`fg`” para ver los procesos que hay en segundo plano y enviarlos a primer plano de la siguiente manera.



```
root@localhost:/var/log
[root@localhost log]# bg
[1]+ watch -n 5 ls -l &
[root@localhost log]# fg %1
```

Por último para ver que procesos están siendo ejecutados en segundo plano utilizaremos el comando “[jobs](#)”

```
root@localhost:/var/log
[root@localhost log]# bg
[1]+ watch -n 5 ls -l &
[root@localhost log]# jobs
[1]+ Detenido watch -n 5 ls -l
[root@localhost log]#
```

Monitoreo de procesos

Para poder ver en tiempo real cuales procesos están utilizando los recursos del sistema usaremos un herramienta que viene desde los UNIX de antaño, pero no por eso deja de ser súper eficiente y útil, se trata del comando “[top](#)”



```
root@localhost:/var/log
top - 15:09:12 up 19 min,  2 users,  load average: 0,00, 0,01, 0,05
Tasks:  95 total,   1 running,  93 sleeping,   1 stopped,   0 zombie
%Cpu(s):  0,0 us,   0,0 sy,   0,0 ni,100,0 id,   0,0 wa,   0,0 hi,   0,0 si,   0,0 st
KiB Mem : 1883804 total, 1603616 free,  118160 used,  162028 buff/cache
KiB Swap:  839676 total,  839676 free,    0 used. 1594876 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
    1 root        20   0 128092   6700  3952  S   0,0   0,4   0:01.16 systemd
    2 root        20   0     0     0     0  S   0,0   0,0   0:00.00 kthreadd
    3 root        20   0     0     0     0  S   0,0   0,0   0:00.03 ksoftirqd/0
    6 root        20   0     0     0     0  S   0,0   0,0   0:00.00 kworker/u2:0
    7 root        rt    0     0     0     0  S   0,0   0,0   0:00.00 migration/0
    8 root        20   0     0     0     0  S   0,0   0,0   0:00.00 rcu_bh
    9 root        20   0     0     0     0  S   0,0   0,0   0:00.34 rcu_sched
   10 root        rt    0     0     0     0  S   0,0   0,0   0:00.00 watchdog/0
   12 root         0 -20     0     0     0  S   0,0   0,0   0:00.00 khelper
   13 root        20   0     0     0     0  S   0,0   0,0   0:00.00 kdevtmpfs
   14 root         0 -20     0     0     0  S   0,0   0,0   0:00.00 netns
   15 root        20   0     0     0     0  S   0,0   0,0   0:00.00 khungtaskd
   16 root         0 -20     0     0     0  S   0,0   0,0   0:00.00 writeback
   17 root         0 -20     0     0     0  S   0,0   0,0   0:00.00 kintegrityd
   18 root         0 -20     0     0     0  S   0,0   0,0   0:00.00 bioset
   19 root         0 -20     0     0     0  S   0,0   0,0   0:00.00 kblockd
   20 root         0 -20     0     0     0  S   0,0   0,0   0:00.00 md
```

Como podrán ver, el comando divide en de 2 partes la pantalla, la superior nos indica datos globales del sistema y debajo hay un listado de los procesos.

El tiempo de “**uptime**” que significa hace cuanto que está encendido el equipo, al ser una virtual es muy poco tiempo en el ejemplo, solamente 19 minutos. Pero en servidores de uso profesional o corporativo pueden llegar a tener años de “**uptime**”, es decir que hace años que no se apagan ¡!

También se puede ver el “**load average**” que se trata de la carga que tiene el CPU, como verán hay 3 números de 2 decimales, el primero es el promedio de “**load average**” en el último minuto, luego 5 minutos y 10 minutos.

Estando en la pantalla de “**Top**”, si oprimimos el botón “**1**” cambiaremos la opción de CPU de 1 a muchos, dependiendo de la configuración del Sistema.

Utilizando las teclas “< o >” cambiaremos la columna mediante la cual están ordenados los procesos.



Existen muchas opciones que podrán visualizar entrando a la pantalla de ayuda con la letra “h” y para salir de la misma deberán usar la letra “q”, también pueden consultar el manual con “man top”.

Otra versión con más opciones es el “htop” pero no viene instalado por default y en situaciones críticas tenemos que estar acostumbrados a poder usar el “top” dado que podemos estar sin internet cuando lo necesitemos.

Por último, la forma de listar procesos que más usaran durante el transcurso de su vida de administradores Linux es el comando “ps” existen muchísimos flags para ver diferentes datos sobre los procesos y los resultados son dramáticamente diferentes de acuerdo a la combinación de los mismos, a continuación veremos algunas opciones interesantes. Para más detalles siempre tienen disponible el “man” de cada comando, es muy importante acostumbrarse ya que es la mejor fuente de información para las tareas de un Admin.

```
[root@localhost log]# ps
  PID TTY          TIME CMD
 2288 pts/0    00:00:00 bash
 2394 pts/0    00:00:00 watch
 2512 pts/0    00:00:00 ps
[root@localhost log]#
```

Una opción interesante para usar con “ps” es el flag “--ppid=” donde le podemos indicar que nos liste todos los procesos que dependen de un determinado PID



```
[root@localhost ~]# ps --ppid=1
PID TTY          TIME CMD
 464 ?            00:00:00 systemd-journal
 481 ?            00:00:00 lvmetad
 491 ?            00:00:00 systemd-udevd
 591 ?            00:00:00 auditd
 610 ?            00:00:00 abrt-d
 611 ?            00:00:00 rngd
 615 ?            00:00:00 abrt-watch-log
 616 ?            00:00:00 polkitd
 617 ?            00:00:00 smartd
 619 ?            00:00:00 systemd-logind
 620 ?            00:00:00 dbus-daemon
 627 ?            00:00:00 chronyd
 630 ?            00:00:00 lsmd
 636 ?            00:00:00 crond
 637 ?            00:00:00 atd
 641 ?            00:00:00 login
 651 ?            00:00:00 firewalld
 667 ?            00:00:00 NetworkManager
 972 ?            00:00:00 rsyslogd
 975 ?            00:00:00 tuned
 979 ?            00:00:00 sshd
2112 ?            00:00:00 master
2213 ?            00:00:00 anacron
2252 ?            00:00:00 abrt-dbus
```

El PID 1 es el llamado systemd, que se trata del primer proceso que levanta el kernel en el proceso de arranque del sistema.

Otra opción muy útil es la “-o” con la cual le podemos pasar un listado de columnas que queremos ver sobre los procesos:



```
[root@localhost ~]# ps -o ppid,pid,uname,comm --ppid=1
PPID  PID USER      COMMAND
    1   464 root    systemd-journal
    1   481 root    lvmetad
    1   491 root    systemd-udev
    1   591 root    auditd
    1   610 root    abrt-d
    1   611 root    rngd
    1   615 root    abrt-watch-log
    1   616 polkitd polkitd
    1   617 root    smartd
    1   619 root    systemd-logind
    1   620 dbus    dbus-daemon
    1   627 chrony  chronyd
    1   630 libstor+ lsmd
    1   636 root    crond
    1   637 root    atd
    1   641 root    login
    1   651 root    firewallld
    1   667 root    NetworkManager
    1   972 root    rsyslogd
    1   975 root    tuned
    1   979 root    sshd
    1  2112 root    master
    1  2213 root    anacron
[root@localhost ~]#
```

Otra opción muy interesante es la que nos permite ordenar los procesos por diferentes columnas “`--sort`”, por ejemplo ordenado por consumo de CPU en orden ascendente (los procesos que tienen mayor consumo abajo):

```
[root@localhost ~]# ps aux --sort=-%cpu
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.3 128092 6704 ?        Ss   16:55   0:01 /usr/lib/systemd/systemd --swi
root         2  0.0  0.0      0     0 ?        S    16:55   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    16:55   0:00 [ksoftirqd/0]
root         6  0.0  0.0      0     0 ?        S    16:55   0:00 [kworker/u2:0]
root         7  0.0  0.0      0     0 ?        S    16:55   0:00 [migration/0]
root         8  0.0  0.0      0     0 ?        S    16:55   0:00 [rcu_bh]
root         9  0.0  0.0      0     0 ?        R    16:55   0:00 [rcu_sched]
root        10  0.0  0.0      0     0 ?        S    16:55   0:00 [watchdog/0]
root        12  0.0  0.0      0     0 ?        S<   16:55   0:00 [khelper]
root        13  0.0  0.0      0     0 ?        S    16:55   0:00 [kdevtmpfs]
root        14  0.0  0.0      0     0 ?        S<   16:55   0:00 [netns]
root        15  0.0  0.0      0     0 ?        S    16:55   0:00 [khungtaskd]
root        16  0.0  0.0      0     0 ?        S<   16:55   0:00 [writeback]
root        17  0.0  0.0      0     0 ?        S<   16:55   0:00 [kintegrityd]
root        18  0.0  0.0      0     0 ?        S<   16:55   0:00 [bioset]
root        19  0.0  0.0      0     0 ?        S<   16:55   0:00 [kblockd]
root        20  0.0  0.0      0     0 ?        S<   16:55   0:00 [md]
root        21  0.0  0.0      0     0 ?        S    16:55   0:00 [kworker/0:1]
root        26  0.0  0.0      0     0 ?        S    16:55   0:00 [kswapd0]
root        27  0.0  0.0      0     0 ?        SN   16:55   0:00 [ksmd]
root        28  0.0  0.0      0     0 ?        SN   16:55   0:00 [khugepaged]
root        29  0.0  0.0      0     0 ?        S    16:55   0:00 [fsnotify_mark]
root        30  0.0  0.0      0     0 ?        S<   16:55   0:00 [crypto]
root        38  0.0  0.0      0     0 ?        S<   16:55   0:00 [kthrotld]
root        40  0.0  0.0      0     0 ?        S<   16:55   0:00 [kmpath_rdacd]
```



Lo mismo que antes, pero en orden descendente:

```
[root@localhost ~]# ps aux --sort=-pcpu
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.3 128092  6704 ?        Ss   16:55   0:01 /usr/lib/systemd/systemd --switched-root --systemd
root         2  0.0  0.0      0     0 ?        S    16:55   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    16:55   0:00 [ksoftirqd/0]
root         6  0.0  0.0      0     0 ?        S    16:55   0:00 [kworker/u2:0]
root         7  0.0  0.0      0     0 ?        S    16:55   0:00 [migration/0]
root         8  0.0  0.0      0     0 ?        S    16:55   0:00 [rcu_bh]
root         9  0.0  0.0      0     0 ?        R    16:55   0:00 [rcu_sched]
root        10  0.0  0.0      0     0 ?        S    16:55   0:00 [watchdog/0]
root        12  0.0  0.0      0     0 ?        S<   16:55   0:00 [khelper]
root        13  0.0  0.0      0     0 ?        S    16:55   0:00 [kdevtmpfs]
root        14  0.0  0.0      0     0 ?        S<   16:55   0:00 [netns]
root        15  0.0  0.0      0     0 ?        S    16:55   0:00 [khungtaskd]
root        16  0.0  0.0      0     0 ?        S<   16:55   0:00 [writeback]
root        17  0.0  0.0      0     0 ?        S<   16:55   0:00 [kintegrityd]
root        18  0.0  0.0      0     0 ?        S<   16:55   0:00 [bioset]
root        19  0.0  0.0      0     0 ?        S<   16:55   0:00 [kblockd]
root        20  0.0  0.0      0     0 ?        S<   16:55   0:00 [md]
root        21  0.0  0.0      0     0 ?        S    16:55   0:00 [kworker/0:1]
root        26  0.0  0.0      0     0 ?        S    16:55   0:00 [kswapd0]
root        27  0.0  0.0      0     0 ?        SN   16:55   0:00 [ksmd]
root        28  0.0  0.0      0     0 ?        SN   16:55   0:00 [khugepaged]
root        29  0.0  0.0      0     0 ?        S    16:55   0:00 [fsnotify_mark]
root        30  0.0  0.0      0     0 ?        S<   16:55   0:00 [crypto]
root        38  0.0  0.0      0     0 ?        S<   16:55   0:00 [kthrotld]
```

También podemos ordenar por utilización de memoria “`--sort=-pmem`”

```
[root@localhost ~]# ps aux --sort=-pmem
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root        651  0.0  1.4 327592 27008 ?        Ss1  16:56   0:00 /usr/bin/python -Es /usr/sbin/firewalld --nofork
root        975  0.0  0.9 553160 18496 ?        Ss1  16:56   0:00 /usr/bin/python -Es /usr/sbin/tuned -l -P
root        771  0.0  0.8 112820 15848 ?        S    16:56   0:00 /sbin/dhclient -d -q -sf /usr/libexec/nm-dhcp-he
polkitd     616  0.0  0.7 530360 13680 ?        Ss1  16:56   0:00 /usr/lib/polkit-1/polkitd --no-debug
root        667  0.0  0.4 513984  8608 ?        Ss1  16:56   0:00 /usr/sbin/NetworkManager --no-daemon
root         1  0.0  0.3 128092  6704 ?        Ss   16:55   0:01 /usr/lib/systemd/systemd --switched-root --systemd
root       2274  0.0  0.2 145184  5540 ?        Ss   17:16   0:00 sshd: root@pts/0
root        610  0.0  0.2 212672  5396 ?        Ss   16:56   0:00 /usr/sbin/abrt-d -d -s
root        491  0.0  0.2  46580  4916 ?        Ss   16:56   0:00 /usr/lib/systemd/systemd-udevd
root        615  0.0  0.2 210344  4592 ?        Ss   16:56   0:00 /usr/bin/abrt-watch-log -F BUG: WARNING: at WARN
root        481  0.0  0.2 195120  4164 ?        Ss   16:56   0:00 /usr/sbin/lvmtools -f
postfix     2127  0.0  0.2  91236  3988 ?        S    16:56   0:00 qmgr -l -t unix -u
postfix     2191  0.0  0.2  91168  3968 ?        S    16:57   0:00 pickup -l -t unix -u
root        972  0.0  0.1 285312  3672 ?        Ss1  16:56   0:00 /usr/sbin/rsyslogd -n
root        641  0.0  0.1 114488  3644 ?        Ss   16:56   0:00 login -- root
root       2278  0.0  0.1 115916  2616 pts/0    Ss   17:16   0:00 -bash
root       2234  0.0  0.1 115912  2572 tty1     Ss+  17:15   0:00 -bash
root        464  0.0  0.1  36816  2448 ?        Ss   16:56   0:00 /usr/lib/systemd/systemd-journald
root        617  0.0  0.1  24324  2296 ?        Ss   16:56   0:00 /usr/sbin/smarterd -n -q never
root       2112  0.0  0.1  91064  2260 ?        Ss   16:56   0:00 /usr/libexec/postfix/master -w
dbus        620  0.0  0.1  34852  2016 ?        Ss1  16:56   0:00 /bin/dbus-daemon --system --address=systemd: ---
root       2325  0.0  0.1 151192  1908 pts/0    R+   17:25   0:00 ps aux --sort=-pmem
chrony      627  0.0  0.0 115848  1848 ?        S    16:56   0:00 /usr/sbin/chronyd
root        591  0.0  0.0  55416  1720 ?        S<sl 16:56   0:00 /sbin/auditd -n
```



Y por último podemos hacer una combinación de las dos cosas

```
[root@localhost ~]# ps aux --sort=+pcpu,-pmem
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      651  0.0  1.4 327592 27008 ?        Ssl   16:56   0:00 /usr/bin/python -Es /usr/sbin/firewalld --nofork
root      975  0.0  0.9 553160 18496 ?        Ssl   16:56   0:00 /usr/bin/python -Es /usr/sbin/tuned -l -P
root      771  0.0  0.8 112820 15848 ?        S     16:56   0:00 /sbin/dhclient -d -q -sf /usr/libexec/nm-dhcp-he
polkitd   616  0.0  0.7 530360 13680 ?        Ssl   16:56   0:00 /usr/lib/polkit-1/polkitd --no-debug
root      667  0.0  0.4 513984 8608 ?        Ssl   16:56   0:00 /usr/sbin/NetworkManager --no-daemon
root       1  0.0  0.3 128092 6704 ?        Ss    16:55   0:01 /usr/lib/systemd/systemd --switched-root --syste
root     2274  0.0  0.2 145184 5540 ?        Ss    17:16   0:00 sshd: root@pts/0
root      610  0.0  0.2 212672 5396 ?        Ss    16:56   0:00 /usr/sbin/abrttd -d -s
root      491  0.0  0.2 46580 4916 ?        Ss    16:56   0:00 /usr/lib/systemd/systemd-udevd
root      615  0.0  0.2 210344 4592 ?        Ss    16:56   0:00 /usr/bin/abrt-watch-log -F BUG: WARNING: at WARN
root      481  0.0  0.2 195120 4164 ?        Ss    16:56   0:00 /usr/sbin/lvmtool -f
postfix   2127  0.0  0.2 91236 3988 ?        S     16:56   0:00 qmgr -l -t unix -u
postfix   2191  0.0  0.2 91168 3968 ?        S     16:57   0:00 pickup -l -t unix -u
root      972  0.0  0.1 285312 3684 ?        Ssl   16:56   0:00 /usr/sbin/rsyslogd -n
root      641  0.0  0.1 114488 3644 ?        Ss    16:56   0:00 login -- root
root     2278  0.0  0.1 115916 2616 pts/0    Ss    17:16   0:00 -bash
root     2234  0.0  0.1 115912 2572 tty1     Ss+   17:15   0:00 -bash
root      464  0.0  0.1 36816 2448 ?        Ss    16:56   0:00 /usr/lib/systemd/systemd-journald
root      617  0.0  0.1 24324 2296 ?        Ss    16:56   0:00 /usr/sbin/smartd -n -q never
root     2112  0.0  0.1 91064 2260 ?        Ss    16:56   0:00 /usr/libexec/postfix/master -w
dbus      620  0.0  0.1 34852 2016 ?        Ssl   16:56   0:00 /bin/dbus-daemon --system --address=systemd: --n
root     2333  0.0  0.1 151192 1904 pts/0    R+    17:30   0:00 ps aux --sort=+pcpu,-pmem
chrony    627  0.0  0.0 115848 1848 ?        S     16:56   0:00 /usr/sbin/chronyd
root      591  0.0  0.0 55416 1720 ?        S<s1  16:56   0:00 /sbin/auditd -n
```

Otro comando muy útil es el “**pstree**” que nos muestra un árbol de dependencias de los procesos para facilitarnos la vida a la hora de encontrar el padre de una serie de procesos. Lamentablemente no viene instalado por default así que lo haremos:

```
[root@localhost log]# yum provides pstree
Complementos cargados:fastestmirror, langpacks
Loading mirror speeds from cached hostfile
* base: centos.xpg.com.br
* extras: centos.xpg.com.br
* updates: centos.xpg.com.br
psmisc-22.20-15.el7.x86_64 : Utilities for managing processes on your system
Repositorio                : base
Resultado obtenido desde:
Nombre del archivo         : /usr/bin/pstree
```

Lo primero es verificar en que paquete de instalación se encuentra el binario para ello utilizaremos el comando “**yum**” que se trata de



nuestro administrador de paquetes y con el flag “**provides**” para buscar que paquete es el que lo contiene.

Una vez que ya sabemos el paquete que tiene el binario que necesitamos lo instalamos con el mismo administrador de paquetes “**yum**”, pero ahora utilizaremos el flag “**install**” seguido del paquete que queremos:

```
[root@localhost log]# yum install psmisc-22.20-15.el7.x86_64
Complementos cargados:fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * base: centos.xpg.com.br
 * extras: centos.xpg.com.br
 * updates: centos.xpg.com.br
Resolviendo dependencias
--> Ejecutando prueba de transacción
--> Paquete psmisc.x86_64 0:22.20-15.el7 debe ser instalado
--> Resolución de dependencias finalizada

Dependencias resueltas

=====
Package                Arquitectura          Versión               Repositorio           Tamaño
=====
Instalando:
psmisc                  x86_64                22.20-15.el7         base                   141 k
=====
```

```
Resumen de la transacción
=====
Instalar 1 Paquete

Tamaño total de la descarga: 141 k
Tamaño instalado: 475 k
Is this ok [y/d/N]: y
Downloading packages:
advertencia:/var/cache/yum/x86_64/7/base/packages/psmisc-22.20-15.el7.x86_64.rpm: EncabezadoV3 RSA/SHA256 Signature, ID de clave f4a80eb5: NOKEY
No se ha instalado la llave pública de psmisc-22.20-15.el7.x86_64.rpm
psmisc-22.20-15.el7.x86_64.rpm | 141 kB 00:00:01
Obteniendo clave desde file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
Importando llave GPG 0xF4A80EB5:
 Usuarioid : "CentOS-7 Key (CentOS 7 Official Signing Key) <security@centos.org>"
 Huella    : 6341 ab27 53d7 8a78 a7c2 7bb1 24c6 a8a7 f4a8 0eb5
 Paquete   : centos-release-7-3.1611.el7.centos.x86_64 (@anaconda)
 Desde     : /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
Está de acuerdo [s/N]:S
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Instalando : psmisc-22.20-15.el7.x86_64 1/1
  Comprobando : psmisc-22.20-15.el7.x86_64 1/1

Instalado:
psmisc.x86_64 0:22.20-15.el7

¡Listo!
[root@localhost log]#
```

Nos preguntará si estamos seguros de que queremos instalar ese paquete y nos dará la opción [s/N]. Una vez que hayamos terminado de instalarlo podremos ejecutarlo!



```
{Listo!
[root@localhost log]# pstree
systemd--NetworkManager--dhclient
                        2*[{NetworkManager}]
--abrt-watch-log
--abrt-d
--atd
--auditd--{auditd}
--chronyd
--crond
--dbus-daemon--{dbus-daemon}
--firewalld--{firewalld}
--login--bash
--lsmd
--lvmetad
--master--pickup
                qmgr
--polkitd--5*[{polkitd}]
--rngd
--rsyslogd--2*[{rsyslogd}]
--smartrd
--sshd--sshd--bash--pstree
                        watch
--systemd-journal
--systemd-logind
--systemd-udevtrd
--tuned--4*[{tuned}]
[root@localhost log]#
```

Administración de Procesos

Como vimos con anterioridad los procesos en Linux pueden encontrarse en diferentes estados durante su vida útil, por lo cual a veces tenemos que administrar el estado de los mismos.

Una de las situaciones más común en la administración de procesos es que, ya sea porque el proceso no está reaccionando, el programa se encuentra colgado, o porque simplemente no queremos que siga en ejecución, necesitamos “matarlo”. Para ello utilizaremos el comando “kill”.



Como podrán ver en la pantalla a continuación, el comando requiere de una de las señales que habíamos mencionado anteriormente.

```
KILL(1) User Commands KILL(1)

NAME
    kill - terminate a process

SYNOPSIS
    kill [-s signal|-p] [-q sigval] [-a] [--] pid...
    kill -l [signal]

DESCRIPTION
    The command kill sends the specified signal to the specified process or process group. If no signal is specified, the TERM signal is sent. The TERM signal will kill processes which do not catch this signal. For other processes, it may be necessary to use the KILL (9) signal, since this signal cannot be caught.

    Most modern shells have a builtin kill function, with a usage rather similar to that of the command described here. The '-a' and '-p' options, and the possibility to specify processes by command name are a local extension.

    If sig is 0, then no signal is sent, but error checking is still performed.

OPTIONS
    pid... Specify the list of processes that kill should signal. Each pid can be one of five things:

        n      where n is larger than 0. The process with pid n will be signaled.

        0      All processes in the current process group are signaled.

        -1     All processes with pid larger than 1 will be signaled.
```

Pero ¿y cuáles son los valores que pueden tomar las señales? Si miramos el flag “-l” podremos ver las posibles señales que podemos enviar a los procesos.

```
[root@localhost log]# kill -l
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
6) SIGABRT     7) SIGBUS      8) SIGFPE      9) SIGKILL     10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE    14) SIGALRM     15) SIGTERM
16) SIGSTKFLT  17) SIGCHLD    18) SIGCONT    19) SIGSTOP     20) SIGTSTP
21) SIGTTIN    22) SIGTTOU    23) SIGURG     24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH   29) SIGIO        30) SIGPWR
31) SIGSYS     34) SIGRTMIN   35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5 60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX

[root@localhost log]#
```

La señal que utilizarán más comúnmente es la 9, SIGKILL que justamente es para matar procesos.



```
[root@localhost log]# jobs
[1]-  Detenido                watch -n 5 ls -l
[2]+  Detenido                watch -n 5 ls -l
[root@localhost log]# bg
[2]+ watch -n 5 ls -l &
[root@localhost log]# ps -ef|grep watch
root      10      2  0 sep25 ?        00:00:00 [watchdog/0]
root      633      1  0 sep25 ?        00:00:00 /usr/bin/abrt-watch-log -F BUG: WARNING: at WARNING:
CPU: INFO: possible recursive locking detected ernel BUG at list_del corruption list_add corruption d
o_IRQ: stack overflow: ear stack overflow (cur: eneral protection fault nable to handle kernel ouble
fault: RTNL: assertion failed eek! page_mapcount(page) went negative! adress at NETDEV WATCHDOG ysc1
table check failed : nobody cared IRQ handler type mismatch Machine Check Exception: Machine check e
vents logged divide error: bounds: coprocessor segment overrun: invalid TSS: segment not present: inv
alid opcode: alignment check: stack segment: fpu exception: simd exception: ired exception: /var/log/
messages -- /usr/bin/abrt-dump-oops -xtD
root      2394    2288  0 sep25 pts/0      00:00:00 watch -n 5 ls -l
root      3999    2288  0 09:37 pts/0      00:00:00 watch -n 5 ls -l
root      4005    2288  0 09:38 pts/0      00:00:00 grep --color=auto watch

[2]+  Detenido                watch -n 5 ls -l
[root@localhost log]# kill -9 3999 2394
[root@localhost log]# ps -ef|grep watch
root      10      2  0 sep25 ?        00:00:00 [watchdog/0]
root      633      1  0 sep25 ?        00:00:00 /usr/bin/abrt-watch-log -F BUG: WARNING: at WARNING:
CPU: INFO: possible recursive locking detected ernel BUG at list_del corruption list_add corruption d
o_IRQ: stack overflow: ear stack overflow (cur: eneral protection fault nable to handle kernel ouble
fault: RTNL: assertion failed eek! page_mapcount(page) went negative! adress at NETDEV WATCHDOG ysc1
table check failed : nobody cared IRQ handler type mismatch Machine Check Exception: Machine check e
vents logged divide error: bounds: coprocessor segment overrun: invalid TSS: segment not present: inv
alid opcode: alignment check: stack segment: fpu exception: simd exception: ired exception: /var/log/
messages -- /usr/bin/abrt-dump-oops -xtD
root      4007    2288  0 09:39 pts/0      00:00:00 grep --color=auto watch
[1]- Terminado (killed)      watch -n 5 ls -l
[2]+ Terminado (killed)      watch -n 5 ls -l
[root@localhost log]#
```

En este ejemplo pueden ver que hay 2 procesos que se están ejecutando en segundo plano en estado detenido.

Luego de ejecutar el “kill -9” seguido de los 2 PID que deseo terminar, al ejecutar nuevamente el “ps -ef” se muestran como terminados. Los procesos van a estar presentes en la tabla de procesos del kernel por un pequeño tiempo y luego serán eliminados.

```
[root@localhost log]#
[root@localhost log]# ps -ef|grep watch
root      10      2  0 sep25 ?        00:00:00 [watchdog/0]
root      633      1  0 sep25 ?        00:00:00 /usr/bin/abrt-watch-log -F BUG: WARNING: at WARNING: CPU: INFO: possible
recursive locking detected ernel BUG at list_del corruption list_add corruption do_IRQ: stack overf
low (cur: eneral protection fault nable to handle kernel ouble fault: RTNL: assertion failed eek! page_mapcount(page) we
nt negative! adress at NETDEV WATCHDOG ysc1 table check failed : nobody cared IRQ handler type mismatch Machine Check E
xception: Machine check events logged divide error: bounds: coprocessor segment overrun: invalid TSS: segment not presen
t: invalid opcode: alignment check: stack segment: fpu exception: simd exception: ired exception: /var/log/messages -- /
usr/bin/abrt-dump-oops -xtD
root      4025    2288  0 09:58 pts/0      00:00:00 grep --color=auto watch
[root@localhost log]#
```

Otra de las formas de administrar procesos en Linux es mediante el cambio de prioridad de los mismos. Esto quiere decir que podemos darles preferencia a unos u otros para acceder a los recursos del



sistema, concretamente al CPU.

Para cambiar la prioridad de un CPU usaremos el comando “**nice**” o “**renice**” asignándole un número del -20 a 19 al proceso que queremos modificar. Siendo -20 el valor más elevado, es decir el de mayor prioridad en cuanto a utilización de CPU.

Se preguntarán “Qué diferencia hay entre ambos comandos?” ... Simple, la diferencia es que “**nice**” se utiliza para “arrancar” un proceso con una determinada prioridad:

```
root@localhost:/var/log
[root@localhost log]# nice -n 5 watch -n 5 ps -ef
```

En cambio el comando “**renice**” se utiliza para cambiar la prioridad de un proceso ya existente.

```
root@localhost:/var/log
[root@localhost log]# nice -n 5 watch -n 5 ps -ef &
[1] 4103
[root@localhost log]# renice +5 4103
4103 (process ID) old priority 5, new priority 5

[1]+  Detenido               nice -n 5 watch -n 5 ps -ef
[root@localhost log]#
```

Para ver la prioridad de los procesos utilizaremos el comando “**ps**” con los flags “**-fl**”

```
[root@localhost log]# ps -fl
F S UID      PID  PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
4 S root       2288 2284  0  80   0 - 29597 wait  sep25 pts/0    00:00:00 -bash
0 T root       4103 2288  0  85   5 - 38237 signal 10:16 pts/0    00:00:00 watch -n 5 ps
0 R root       4184 2288  0  80   0 - 37764 -      11:34 pts/0    00:00:00 ps -fl
[root@localhost log]#
```




Administración de Servicios

Anteriormente mencionamos el proceso con PID 1 llamado “**systemd**”, dijimos que este proceso arranca una serie de servicios del sistema operativo de los cuales hacemos uso normalmente, tanto como usuarios comunes o como administradores del sistema.

Además de iniciar los servicios al arranque del sistema, “**systemd**” nos permite administrarlos de una forma muy conveniente y sencilla.

Inicio y Detención de Servicios

Para seguir adelante vamos a necesitar un servicio que podamos arrancar y detener sin que nos genere inconvenientes en el sistema, por lo cual vamos a tener que instalar un servidor web, para ello utilizaremos el comando “**yum**” como hemos hecho anteriormente.

Primero buscaremos los paquetes que necesitamos y luego los instalaremos:

```
[root@localhost ~]# yum search httpd
Complementos cargados:fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * base: centos.xpg.com.br
 * extras: centos.xpg.com.br
 * updates: centos.xpg.com.br

===== N/S matched: httpd =====
keycloak-httpd-client-install.noarch : Tools to configure Apache HTTPD as Keycloak client
libmicrohttpd-devel.i686 : Development files for libmicrohttpd
libmicrohttpd-devel.x86_64 : Development files for libmicrohttpd
libmicrohttpd-doc.noarch : Documentation for libmicrohttpd
python2-keycloak-httpd-client-install.noarch : Tools to configure Apache HTTPD as Keycloak client
httpd.x86_64 : Apache HTTP Server
httpd-devel.x86_64 : Development interfaces for the Apache HTTP server
httpd-manual.noarch : Documentation for the Apache HTTP server
httpd-tools.x86_64 : Tools for use with the Apache HTTP Server
libmicrohttpd.i686 : Lightweight library for embedding a webserver in applications
libmicrohttpd.x86_64 : Lightweight library for embedding a webserver in applications
mod_auth_mellon.x86_64 : A SAML 2.0 authentication module for the Apache Httpd Server
mod_dav_svn.x86_64 : Apache httpd module for Subversion server

Nombre y resumen que coinciden con y sólo , use "buscar todo" para todo.
[root@localhost ~]# yum install httpd.x86_64 httpd-manual.noarch
```

Necesitamos el paquete de “**httpd**” y su respectivo manual online para poder verlo. Como siempre nos consultará si estamos de acuerdo con los paquetes a instalar e ingresamos “**y**”.

Listo, ya terminamos de instalar nuestro servidor web Apache (así



es como se llama el proyecto open source)

Ahora es momento de empezar a administrarlo como un servicio.

Para la manipulación de los servicios utilizaremos el comando “**systemctl**” seguido de algunas opciones muy sencillas:

```
[root@localhost ~]# systemctl start httpd
[root@localhost ~]# systemctl stop httpd
[root@localhost ~]#
```

Con “**start**” seguido del nombre del servicio lo iniciamos y con “**stop**” lo detenemos.

Pero como sabemos si realmente está funcionando? Para eso existe la opción “**status**”.

```
[root@localhost ~]# systemctl status httpd
• httpd.service - The Apache HTTP Server
  Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset: disabled)
  Active: inactive (dead)
  Docs: man:httpd(8)
        man:apachectl(8)

sep 27 18:25:01 localhost.localdomain systemd[1]: Starting The Apache HTTP Server...
sep 27 18:25:01 localhost.localdomain httpd[2523]: AH00558: httpd: Could not reliably determine the server's fully...ssage
sep 27 18:25:01 localhost.localdomain systemd[1]: Started The Apache HTTP Server.
sep 27 18:25:49 localhost.localdomain systemd[1]: Stopping The Apache HTTP Server...
sep 27 18:25:50 localhost.localdomain systemd[1]: Stopped The Apache HTTP Server.
Hint: Some lines were ellipsized, use -l to show in full.
[root@localhost ~]#
```

Como era de esperarse el sistema me indica que el servicio se encuentra detenido.

Una opción muy importante es la de habilitar un servicio, esto se hace mediante la opción “**enable**” y significa que el servicio arrancará automáticamente una vez que reiniciemos el sistema.

Por el contrario la opción “**disable**” elimina del “listado de arranque” al servicio para que no se inicie cada vez que reiniciamos el sistema por lo cual deberemos hacerlo manualmente en caso de desearlo.

Por último, para poder ver la lista de servicios disponibles se ejecuta el comando “**systemctl**”, dado que la lista es largo pueden combinarlo con el comando “**grep**” utilizando “**|**” como hemos visto



anteriormente.

```
[root@localhost ~]#  
[root@localhost ~]# systemctl|grep sshd  
sshd.service loaded activ  
e running OpenSSH server daemon  
[root@localhost ~]#
```

Volviendo a nuestro caso de estudio, vamos a dejarlo habilitado para poder utilizarlo sin tener que estar iniciándolo cada vez que reiniciemos el sistema

```
[root@localhost ~]#  
[root@localhost ~]# systemctl enable httpd  
Created symlink from /etc/systemd/system/multi-user.target.wants/httpd.service to /usr/lib/systemd/system/httpd.service.  
[root@localhost ~]#
```

Vamos nuevamente a iniciarlo para poder comprobar que nuestro Apache se encuentra funcionando correctamente.

Lo que haremos ahora es probar si podemos acceder con nuestro navegador web, ya sea Internet Explorer, Firefox o Chrome. Para ello deben abrirlo y en la barra de navegación poner la IP que tienen en su instancia de máquina virtual con Linux, la misma que pusieron en putty para poder acceder a la misma vía ssh.



Ups ¡! Que está pasando ??? Nos olvidamos que Linux es un sistema operativo seguro! Eso implica que viene con un firewall instalado y que debemos habilitar el puerto donde está escuchando el

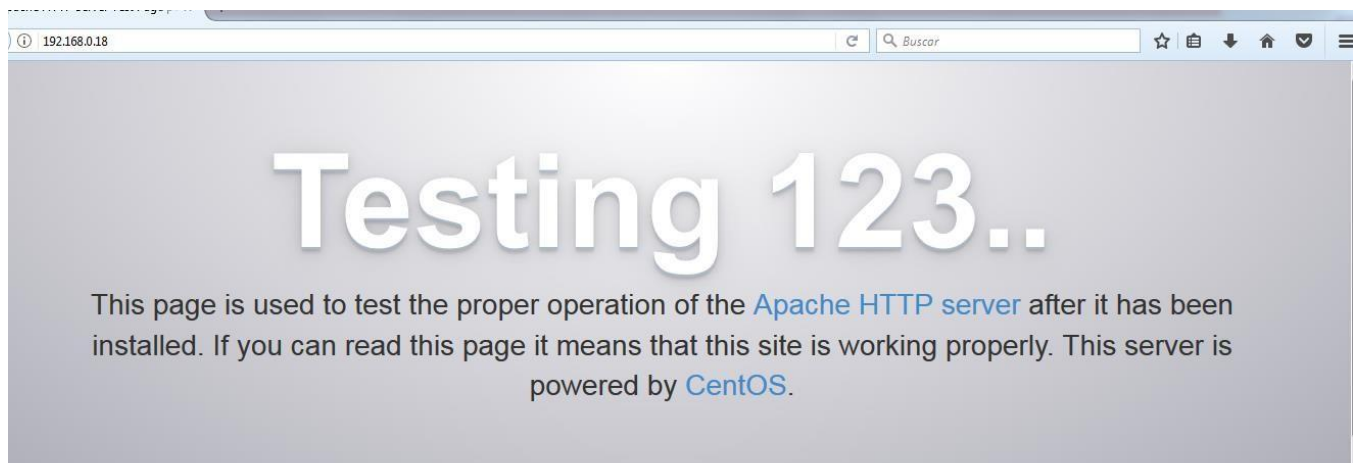


servidor apache (puerto 80) , para ello vamos a ejecutar el siguiente comando: “[firewall-cmd --zone=public --add-port=80/tcp --permanent](#)”
Con esto le decimos al firewall que queremos recibir comunicaciones en el puerto indicado y que sea persistente , es decir que sobreviva a un reboot.

Luego tenemos que reiniciar el firewall con el siguiente comando:
“[firewall-cmd -reload](#)”

```
root@localhost:~  
[root@localhost ~]# firewall-cmd --zone=public --add-port=80/tcp --permanent  
success  
[root@localhost ~]# firewall-cmd --reload  
success  
[root@localhost ~]#
```

Listo ¡! Ahora si podemos probar nuestro servidor web ¡!



Excelente ¡! Se acuerdan del manual que instalamos? Bueno pueden acceder ingresando con el navegador a su ip y luego agregando “[/manual](#)” en la barra de direcciones



192.168.0.18/manual/ Buscar

Modules | Directives | FAQ | Glossary | Sitemap

Apache HTTP Server Version 2.4

Apache > HTTP Server > Documentation

Apache HTTP Server Version 2.4 Documentation

Available Languages: da | de | en | es | fr | ja | ko | pt-br | tr | zh-cn

Google Search

Release Notes New features with Apache 2.3/2.4 New features with Apache 2.1/2.2 New features with Apache 2.0 Upgrading to 2.4 from 2.2 Apache License	Users' Guide Binding to Addresses and Ports Configuration Files Configuration Sections Content Caching Content Negotiation Dynamic Shared Objects (DSO) Environment Variables Log Files Mapping URLs to the Filesystem Performance Tuning Security Tips Server-Wide Configuration SSL/TLS Encryption Suexec Execution for CGI URL Rewriting with mod_rewrite Virtual Hosts	How-To / Tutorials Authentication and Authorization Access Control CGI: Dynamic Content .htaccess files Server Side Includes (SSI) Per-user Web Directories (public_html)
Reference Manual Compiling and Installing Starting Stopping or Restarting Run-time Configuration Directives Directive Quick-Reference Modules Multi-Processing Modules (MPMs) Filters Handlers Expression parser Server and Supporting Programs		Platform Specific Notes Microsoft Windows RPM-based Systems (Redhat / CentOS / Fedora) Novell NetWare EBCDIC Port
		Other Topics Frequently Asked Questions Sitemap Documentation for Developers Helping with the documentation

Ejercicio

- 1) Instalar el paquete MariaDB utilizando el comando “yum”.
- 2) Configurar MariaDB como servicio habilitado (que se inicie automáticamente luego de un reboot) e iniciar el servicio.
- 3) Verificar que el servicio esté ejecutándose, ya sea mediante la utilidad provista por “systemd” o listando los procesos.
- 4) Abrir el puerto de firewall necesario para poder acceder a la base de datos MariaDB remotamente.

Links Recomendados

<http://httpd.apache.org/docs/>

<https://mariadb.com/kb/en/library/documentation/>

<https://access.redhat.com/documentation/en/red-hat-enterprise-linux/>