ps : Muestra el estado de los procesos. Por defecto *ps* solo muestra los procesos que se ejecutaron desde su propia terminal (xterm, acceso en modo texto o acceso remoto).

$ps [-|–] OPCIONES

[Tipos de opciones]:

- Opciones Unix98 : Opción de un único caracter, precedido por "-". Se pueden agrupar ($ps -e-f = ps-ef)
- Opciones BSD : Opción de un único caracter, pero no van precedido de "-". También se pueden agrupar
- Opciones GNU largas: Opciones multicaracter y precedidas de "–"

[Opciones Unix98]:

- Mostrar opciones básicas (PID, TTY, TIME, CMD): -A , -e
- Mostrar los procesos que pertenecen a un usuario concreto bien por su ID o por el nombre de login: -u usuario
- Mostrar información adicional: -f, -l . Podemos usar -L con -f para ver el número e ID de hilo
- Mostrar la jerarquía de procesos, útil para averiguar los parentescos de un proceso: -H -f ó –forest (agrupan los procesos y emplean sangrías)
- Mostrar una salida ancha, útil si queremos redireccionarla a un archivo para luego examinarla (por defecto ps recorta la salida a 80 columnas) : -w > archivo.txt

[Opciones BSD]:

- Mostrar todos los procesos del usuario que proporciona el comando : x
- Ver los procesos de un usuario concreto: U usuario
- Mostrar información adicional: j (formato de control de trabajos), l (formato largo de BSD), u (formato orientado al usuario) y v (formato de memoria virtual)
- No recortar la salida: w

[Opciones GNU]:

- Mostrar la ayuda de ps: –help
- Mostrar procesos de un usuario: –User usuario
- Mostrar la jerarquía de procesos: –forest

Algunos ejemplos:

- Ver todos los procesos del sistema:

```
ps -ef, ps -eF, ps -ely, ps ax, ps axu
```

- Imprimir el árbol de procesos:

```
ps -ejH, ps axjf
```

- Mostrar información sobre los hilos:

```
ps -eLf, ps axms
```

- Mostrar información de seguridad:

```
ps -eM, ps axZ
```

- Mostrar los números ID's de un programa concreto:

```
ps -C nombre_prog -o pid=
```

- Mostrar el nombre de programa de un proceso:

```
ps -p num_pid -o comm=
```

pgrep : Herramienta que combina los comandos *ps* y *grep*, para imprimir los ID de procesos que coinciden con el patrón pasado en la expresión regular.

$pgrep [opciones] expresión regular [archivo]

Ejemplo:

```
$ps ax |grep http | grep -v | awk '{print $1}' == $pgrep http
```

[Opciones]:

- Imprimir los procesos ejecutados por un determinado usuario: **-u**
- Mostrar el nombre del proceso: **-l**
- Mostrar todos los procesos que no coincidan con el patrón: **-v**
- Imprimir la ruta del binario del proceso: **-f**
- Mostrar el total de los procesos que coinciden con el patrón mediante un dígito: **-c**
- Mostrar los procesos de un determinado grupo mediante el nombre del grupo: **-G** (-g para el GID)
- Identificar el proceso mas viejo: **-o**
- Identificar el proceso mas nuevo: **-n**

Referencias:

http://es.wikipedia.org/wiki/Pgrep

http://systemadmin.es/2008/12/pkill-y-pgrep-buscar-o-mandar-senales-por-el-nombre-del-proceso

http://linux.floresdecerezo.com/el-comando-pgrep/

fuser: Nos permite identificar que procesos están haciendo uso determinados archivos, *sockets* o directorios, o dicho de otra manera por que procesos están siendo controlados determinados archivos. También nos permite matar procesos.

fuser [opciones] archivo

Opciones:

- Mostrar los procesos para todos los archivos pasados desde la línea de comandos: -a, –all
- Mata un proceso: -k
- Activa el modo interactivo: -i
- Muestra mas información: -v
- Mostrar todas las señales conocidas -l
- Matar solo procesos que tienen acceso de escritura: -w
- Activar el modo silencioso: -s
- Añade el nombre del usuario propietario para cada PID: -u, –user

top : Muestra un resumen de la información del sistema y de los procesos del ordenador que mas CPU consumen.

$top -opciones

[Opciones]:

- Especificar un retardo diferente al de por defecto (5s) para la actualización de la info: **-d**
- Monitorizar procesos específicos (hasta 20): **-p** *PID*
- Mostrar un número concreto de actualizaciones y luego cerrarse: **-n** *num_iteraciones*
- Utilizar un archivo para recoger el uso de CPU de determinados programas: **-b**

[Teclas para interaccionar con top]:

- Mostrar información de ayuda: **h**, **y**, **?**
- Destruir un proceso conociendo su PID: **k**
- Cambiar la prioridad de un proceso: **r**
- Modificar el intervalo de refresco de información: **s**
- Ordenar los datos mostrados por uso de CPU (top por defecto actúa así): **P**
- Ordenar los datos por uso de memoria: **M**
- Salir de top: **q**

uptime : Nos permite conocer el tiempo que lleva el sistema sin ser reiniciado, así como la carga media (load average).

$uptime

Nota: El comando *uptime* desplegará una línea con 4 campos: la hora actual, el tiempo que lleva el sistema iniciado sin ser reiniciado, el número de usuarios conectados y el promedio de carga del sistema en los últimos 1, 5 y 15 minutos.

También podemos ver el tiempo sin reinicio del sistema en segundos desplegando *$cat /proc/uptime* o el promedio de carga a través del archivo */proc/loadavg*

free : Muestra la cantidad de memoria física y de intercambio libre y usada en el sistema, así como el búfer utilizado por el kernel.

$free [opciones]

[Opciones]:

- Mostrar la memoria en bytes, kilobytes, megabytes o gigabytes: **-b -k -m -g** (*respectivamente*)
- Mostrar la cantidad de memoria en unidades (B, M, K, G o T) directamente abreviadas según la cantidad: **-h**
- Mostrar la salida del comando durante un tiempo deseado: **-s** <*num_seg*>
- Mostrar la salida de *free* tantas veces como deseemos. Es necesario acompañarlo de la opción s: **-c** <*num_veces*>

jobs : Identifica el número de trabajos en ejecución. Útil para conocer los ID de tareas y pasar estas de primer a segundo plano, suspenderlas o terminarlas

$jobs

& : Si lo añadimos al final de la línea de ejecución del comando nos permitirá que este se ejecute en segundo plano, dejando libre la terminal.

wait : Nos permite elegir cuando queremos retomar el control de la terminal en el caso de que haya varios programas en background.

$wait

[Opciones]:

- Si queremos esperar a que terminen todos los trabajos, no pasaremos ninguna opción.
- Si queremos retomar el control cuando termine el trabajo [2]: **%**2
- Si queremos retomar el control cuando termine el proceso 4563: <num_proc>

nohup : Permite la continuidad en la ejecución de un programa/comando aun habiendo cerrado la terminal desde la que se ejecutó.

$nohup [comando/programa] [opciones comando/programa]

screen : Crea terminales virtuales para poder ejecutar un comando pudiendo cerrar la ventana y que este siga ejecutándose, además poder abrir de nuevo esa misma terminal virtual y retomar el control.

$screen [opciones]

[Opciones]:

- Si escribimos 'screen' a secas, abriremos una nueva terminal virtual
- Si queremos listar las terminales virtuales existentes en el equipo: **-ls**
- Para retomar el control de una determinada terminal virtual: **-r** *ID*

Nota: Si solo hay una terminal virtual abierta, basta con escribir screen *-r*

nice : El comando *nice* nos permite ejecutar programas con una determinada prioridad en la CPU.

$nice [prioridad] [programa] [argumentos del programa]

```
$nice -12 [programa] [argumentos...]

$nice -n 12 [programa] [argumentos...]

$nice --adjustment=12 [programa] [argumentos...]
```

Nota: Estos 3 comando ejecutan "programa" con prioridad 12 positiva. Para prioridades negativas solo podemos usar las 2 últimas. Si no le pasamos prioridad a *nice*, inicia el programa con prioridad 10

[Prioridad]:

- La prioridad va desde -20 a 19, siendo los números negativos los de prioridad mas elevada y solo pueden se utilizados por root.

renice : Permite modificar la prioridad de un programa en ejecución. La sintaxis es igual a la de *nice*, pero además nos permite modificar la prioridad de programa(s) por PIDs, por GUIDs y por nombres de usuarios

$renice [prioridad] [-p PIDs | -g grps | -u usuarios] [programa] [argumentos…]

Nota: Solo *root* puede modificar la prioridad por GUIs y nombre de usuarios. Si no se le pasa prioridad a *renice*, entiende que el primer número es un *PID*.

kill : Envía una señal al kernel para finalizar un proceso

$kill [parametro] [señal]

[Parámetro]:

- Para indicar la señal por su nombre completo: **-s**
- Para indica la señal por su nombre (sin SIG): **–***señal*
- Para pasarle el número de señal: **–***num*

Nota: El comando *kill* también acepta el parámetro *%id_job* para pasar una señal a determinado trabajo. **$kill %3**

pkill : Es igual que *pgrep*, pero además mata el proceso, es decir, se le pasa un nombre de programa como patrón, un usuario o grupo de usuarios y mata estos procesos. Además permite pasar señales a procesos.

$pkill [opciones] expresión regular

Ejemplo:

```
$pkill -HUP syslog
```

Nota: Este comando le pasa la señal de finalización a *syslog*

[Opciones]:

- Aquellas que se le pasen para identificar a un proceso, serán las que sirvan para identificar y matarlo.
- Enviar una determinada señal al proceso: **–**SEÑAL (sin SIG)

killall : Nos permite matar un proceso por su nombre y no por su *PID*. También podemos pasarles señales como a *kill*. Existe una variante de Unix que mata todos los procesos iniciados por un determinado usuario, en caso de pasarle como argumento el nombre del usuario.

```
$killall -i vi
```

[Opciones]:

- Para pedir confirmación de finalización de proceso: **-i**
- Todos los procesos de un determinado usuario: **-u**
- Todos los procesos de un determinado grupo: **-g**
- Incluir expresión regular para la búsqueda de los procesos: **-r**
- Mandar una señal de finalización: **-s**