

Evaluación: ps, grep, pipes linux, bash, awk

Objetivos

- Aprender a listar y filtrar procesos activos en un sistema.
- Entender cómo identificar procesos por PID, usuario, uso de recursos y otros criterios.
- Utilizar **ps** para monitorear la salud y el rendimiento de aplicaciones paralelas y distribuidas.
- Aplicar **grep** para analizar logs de aplicaciones y sistemas, facilitando la depuración y el monitoreo.
- Utilizar pipes para crear cadenas de procesamiento de datos eficientes y scripts de análisis
- Aprender a escribir scripts de shell para automatizar tareas de administración y despliegue.
- Entender el control de flujo, manejo de variables, y funciones en Bash.
- Desarrollar habilidades para la automatización de pruebas y despliegues en entornos de computación distribuida.
- Aprender a utilizar **awk** para el filtrado y transformación de datos complejos en scripts de shell.

Entregable:

Presenta el código completo y tus respuestas desarrollado en tu repositorio personal hasta el día 16 de abril (8:00 PM). Recuerda presentar tus resultados en formato markdown y código si es que se ha realizado.

El comando ps

El comando **ps** en Linux y otros sistemas tipo Unix es una herramienta de línea de comandos utilizada para mostrar información sobre los procesos activos en un sistema. **ps** es el acrónimo de "process status" o estado del proceso. Proporciona una instantánea de los procesos corriendo en ese momento, incluyendo detalles como el ID del proceso (PID), el usuario propietario del proceso, el uso de CPU, el uso de memoria, el tiempo de ejecución, el comando que inició el proceso, entre otros.

En un curso de computación paralela, concurrente y distributiva, el comando **ps** puede ser aplicado de diversas maneras para facilitar la comprensión y gestión de los procesos y la ejecución de programas en estos entornos:

- **Monitoreo de procesos:** **ps** puede ser usado para enseñar cómo identificar y monitorear procesos individuales o grupos de procesos relacionados con aplicaciones paralelas y concurrentes.
- **Gestión de recursos:** Utilizando **ps** junto con otras herramientas, se puede enseñar a observar el uso de CPU y memoria, lo cual es crucial para la optimización de aplicaciones en entornos paralelos y distribuidos.
- **Depuración y diagnóstico:** En la computación paralela y concurrente, identificar procesos bloqueados, zombies o que consumen recursos excesivamente es fundamental para la depuración y el mantenimiento del rendimiento del sistema. **ps** permite identificar rápidamente tales procesos.
- **Automatización y scripting:** **ps** se puede usar en scripts de shell para automatizar la supervisión y gestión de aplicaciones paralelas y distribuidas.
- **Estudio de casos:** Análisis de casos de estudio donde se requiere la identificación y gestión de procesos en sistemas de computación distribuida. Por ejemplo, cómo gestionar de manera eficiente múltiples instancias de un servicio web distribuido en un cluster de servidores.

Ejercicios

1. Listar todos los procesos con detalles completos

ps -ef procesos con detalles completos

```
alumno@administrador-20VE:~$ ps -ef
UID          PID    PPID  C  STIME TTY          TIME CMD
root         1      0  0  11:16 ?        00:00:01 /sbin/init splash
root         2      0  0  11:16 ?        00:00:00 [kthreadd]
root         3      2  0  11:16 ?        00:00:00 [rcu_gp]
root         4      2  0  11:16 ?        00:00:00 [rcu_par_gp]
root         5      2  0  11:16 ?        00:00:00 [slub_flushwq]
root         6      2  0  11:16 ?        00:00:00 [netns]
root         8      2  0  11:16 ?        00:00:00 [kworker/0:0H-events_highpri]
root        10      2  0  11:16 ?        00:00:00 [mm_percpu_wq]
root        11      2  0  11:16 ?        00:00:00 [rcu_tasks_kthread]
root        12      2  0  11:16 ?        00:00:00 [rcu_tasks_rude_kthread]
root        13      2  0  11:16 ?        00:00:00 [rcu_tasks_trace_kthread]
root        14      2  0  11:16 ?        00:00:00 [ksoftirqd/0]
root        15      2  0  11:16 ?        00:00:05 [rcu_preempt]
root        16      2  0  11:16 ?        00:00:00 [migration/0]
root        17      2  0  11:16 ?        00:00:00 [idle_inject/0]
root        19      2  0  11:16 ?        00:00:00 [cpuhp/0]
root        20      2  0  11:16 ?        00:00:00 [cpuhp/1]
root        21      2  0  11:16 ?        00:00:00 [idle_inject/1]
root        22      2  0  11:16 ?        00:00:00 [migration/1]
root        23      2  0  11:16 ?        00:00:00 [ksoftirqd/1]
root        25      2  0  11:16 ?        00:00:00 [kworker/1:0H-events_highpri]
root        26      2  0  11:16 ?        00:00:00 [cpuhp/2]
root        27      2  0  11:16 ?        00:00:00 [idle_inject/2]
root        28      2  0  11:16 ?        00:00:00 [migration/2]
root        29      2  0  11:16 ?        00:00:00 [ksoftirqd/2]
root        31      2  0  11:16 ?        00:00:00 [kworker/2:0H-events_highpri]
root        32      2  0  11:16 ?        00:00:00 [cpuhp/3]
root        33      2  0  11:16 ?        00:00:00 [idle_inject/3]
root        34      2  0  11:16 ?        00:00:00 [migration/3]
root        35      2  0  11:16 ?        00:00:00 [ksoftirqd/3]
root        37      2  0  11:16 ?        00:00:00 [kworker/3:0H-events_highpri]
root        38      2  0  11:16 ?        00:00:00 [cpuhp/4]
root        39      2  0  11:16 ?        00:00:00 [idle_inject/4]
root        40      2  0  11:16 ?        00:00:00 [migration/4]
root        41      2  0  11:16 ?        00:00:00 [ksoftirqd/4]
root        43      2  0  11:16 ?        00:00:00 [kworker/4:0H-events_highpri]
root        44      2  0  11:16 ?        00:00:00 [cpuhp/5]
root        45      2  0  11:16 ?        00:00:00 [idle_inject/5]
root        46      2  0  11:16 ?        00:00:00 [migration/5]
root        47      2  0  11:16 ?        00:00:01 [ksoftirqd/5]
root        49      2  0  11:16 ?        00:00:00 [kworker/5:0H-events_highpri]
root        50      2  0  11:16 ?        00:00:00 [cpuhp/6]
root        51      2  0  11:16 ?        00:00:00 [idle_inject/6]
```

ps aux

```

alumno@administrador-20VE:~$ ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0 168828 13004 ?        Ss   11:16   0:01 /sbin/init sp
root         2  0.0  0.0      0     0 ?        S    11:16   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        I<   11:16   0:00 [rcu_gp]
root         4  0.0  0.0      0     0 ?        I<   11:16   0:00 [rcu_par_gp]
root         5  0.0  0.0      0     0 ?        I<   11:16   0:00 [slub_flushwq
root         6  0.0  0.0      0     0 ?        I<   11:16   0:00 [netns]
root         8  0.0  0.0      0     0 ?        I<   11:16   0:00 [kworker/0:0H
root        10  0.0  0.0      0     0 ?        I<   11:16   0:00 [mm_percpu_wq
root        11  0.0  0.0      0     0 ?        I    11:16   0:00 [rcu_tasks_kt
root        12  0.0  0.0      0     0 ?        I    11:16   0:00 [rcu_tasks_ru
root        13  0.0  0.0      0     0 ?        I    11:16   0:00 [rcu_tasks_tr
root        14  0.0  0.0      0     0 ?        S    11:16   0:00 [ksoftirqd/0]
root        15  0.0  0.0      0     0 ?        I    11:16   0:06 [rcu_preempt]
root        16  0.0  0.0      0     0 ?        S    11:16   0:00 [migration/0]
root        17  0.0  0.0      0     0 ?        S    11:16   0:00 [idle_inject/
root        19  0.0  0.0      0     0 ?        S    11:16   0:00 [cpuhp/0]
root        20  0.0  0.0      0     0 ?        S    11:16   0:00 [cpuhp/1]
root        21  0.0  0.0      0     0 ?        S    11:16   0:00 [idle_inject/
root        22  0.0  0.0      0     0 ?        S    11:16   0:00 [migration/1]
root        23  0.0  0.0      0     0 ?        S    11:16   0:00 [ksoftirqd/1]
root        25  0.0  0.0      0     0 ?        I<   11:16   0:00 [kworker/1:0H
root        26  0.0  0.0      0     0 ?        S    11:16   0:00 [cpuhp/2]
root        27  0.0  0.0      0     0 ?        S    11:16   0:00 [idle_inject/
root        28  0.0  0.0      0     0 ?        S    11:16   0:00 [migration/2]
root        29  0.0  0.0      0     0 ?        S    11:16   0:00 [ksoftirqd/2]
root        31  0.0  0.0      0     0 ?        I<   11:16   0:00 [kworker/2:0H
root        32  0.0  0.0      0     0 ?        S    11:16   0:00 [cpuhp/3]
root        33  0.0  0.0      0     0 ?        S    11:16   0:00 [idle_inject/
root        34  0.0  0.0      0     0 ?        S    11:16   0:00 [migration/3]
root        35  0.0  0.0      0     0 ?        S    11:16   0:00 [ksoftirqd/3]
root        37  0.0  0.0      0     0 ?        I<   11:16   0:00 [kworker/3:0H
root        38  0.0  0.0      0     0 ?        S    11:16   0:00 [cpuhp/4]
root        39  0.0  0.0      0     0 ?        S    11:16   0:00 [idle_inject/
root        40  0.0  0.0      0     0 ?        S    11:16   0:00 [migration/4]
root        41  0.0  0.0      0     0 ?        S    11:16   0:00 [ksoftirqd/4]
root        43  0.0  0.0      0     0 ?        I<   11:16   0:00 [kworker/4:0H
root        44  0.0  0.0      0     0 ?        S    11:16   0:00 [cpuhp/5]
root        45  0.0  0.0      0     0 ?        S    11:16   0:00 [idle_inject/
root        46  0.0  0.0      0     0 ?        S    11:16   0:00 [migration/5]
root        47  0.0  0.0      0     0 ?        S    11:16   0:01 [ksoftirqd/5]
root        49  0.0  0.0      0     0 ?        I<   11:16   0:00 [kworker/5:0H
root        50  0.0  0.0      0     0 ?        S    11:16   0:00 [cpuhp/6]
root        51  0.0  0.0      0     0 ?        S    11:16   0:00 [idle_inject/

```

2. Buscar procesos específicos por nombre:

ps -ef | grep firefox

```

alumno@administrador-20VE:~$ ps -ef | grep firefox
alumno  3341    2055 10 11:20 ?        00:17:17 /snap/firefox/4090/usr/lib/f
firefox/firefox
alumno  3715    3341  0 11:20 ?        00:00:00 /snap/firefox/4090/usr/lib/f
firefox/firefox -contentproc -parentBuildID 20240401184549 -prefsLen 31405 -prefM
apSize 237467 -appDir /snap/firefox/4090/usr/lib/firefox/browser {b220bc97-8d8e-
441b-bc53-20f06ae5a9e5} 3341 true socket
alumno  3740    3341  0 11:20 ?        00:00:03 /snap/firefox/4090/usr/lib/f
firefox/firefox -contentproc -childID 1 -isForBrowser -prefsLen 31546 -prefMapSiz
e 237467 -jsInitLen 234952 -parentBuildID 20240401184549 -greomni /snap/firefox/
4090/usr/lib/firefox/omni.ja -appomni /snap/firefox/4090/usr/lib/firefox/browser
/omni.ja -appDir /snap/firefox/4090/usr/lib/firefox/browser {e0558f00-6835-47cb-
907f-4905ced531e0} 3341 true tab
alumno  3878    3341  0 11:20 ?        00:00:00 /snap/firefox/4090/usr/lib/f
firefox/firefox -contentproc -childID 2 -isForBrowser -prefsLen 36952 -prefMapSiz
e 237467 -jsInitLen 234952 -parentBuildID 20240401184549 -greomni /snap/firefox/
4090/usr/lib/firefox/omni.ja -appomni /snap/firefox/4090/usr/lib/firefox/browser
/omni.ja -appDir /snap/firefox/4090/usr/lib/firefox/browser {a31c4393-f1c1-43f5-
87d0-d2faee792510} 3341 true tab
alumno  4051    3341  0 11:20 ?        00:00:00 /snap/firefox/4090/usr/lib/f
firefox/firefox -contentproc -parentBuildID 20240401184549 -sandboxingKind 0 -pre
fsLen 37063 -prefMapSize 237467 -appDir /snap/firefox/4090/usr/lib/firefox/brows
er {b5e0e2f4-ad9e-4ff7-943a-519abcea1c59} 3341 true utility
alumno  4058    3341  0 11:20 ?        00:01:02 /snap/firefox/4090/usr/lib/f
firefox/firefox -contentproc -childID 3 -isForBrowser -prefsLen 31297 -prefMapSiz
e 237467 -jsInitLen 234952 -parentBuildID 20240401184549 -greomni /snap/firefox/
4090/usr/lib/firefox/omni.ja -appomni /snap/firefox/4090/usr/lib/firefox/browser
/omni.ja -appDir /snap/firefox/4090/usr/lib/firefox/browser {bdb64998-cfd4-4aed-
8909-4eb5c1a5aaf6} 3341 true tab
alumno  4084    3341  3 11:20 ?        00:06:05 /snap/firefox/4090/usr/lib/f
firefox/firefox -contentproc -childID 5 -isForBrowser -prefsLen 31297 -prefMapSiz
e 237467 -jsInitLen 234952 -parentBuildID 20240401184549 -greomni /snap/firefox/
4090/usr/lib/firefox/omni.ja -appomni /snap/firefox/4090/usr/lib/firefox/browser
/omni.ja -appDir /snap/firefox/4090/usr/lib/firefox/browser {89d6d42d-ba59-49e8-
beb3-6734ec9e03bd} 3341 true tab
alumno  4209    3341  3 11:20 ?        00:05:45 /snap/firefox/4090/usr/lib/f
firefox/firefox -contentproc -childID 7 -isForBrowser -prefsLen 31435 -prefMapSiz
e 237467 -jsInitLen 234952 -parentBuildID 20240401184549 -greomni /snap/firefox/
4090/usr/lib/firefox/omni.ja -appomni /snap/firefox/4090/usr/lib/firefox/browser
/omni.ja -appDir /snap/firefox/4090/usr/lib/firefox/browser {9a464a3b-7070-46a1-
8b51-f8df49bfcca6} 3341 true tab
alumno  4335    3341  0 11:20 ?        00:00:00 /snap/firefox/4090/usr/lib/f
firefox/firefox -contentproc -parentBuildID 20240401184549 -prefsLen 37201 -prefM
apSize 237467 -appDir /snap/firefox/4090/usr/lib/firefox/browser {bd292b3e-bf12-
444a-b636-8915e847cd36} 3341 true rdd

```

```
alumno@administrador-20VE:~$ ps aux | grep apache
alumno      9163  0.0  0.0  6576  2432 pts/1    S+   14:15   0:00 grep --color=
auto apache
alumno@administrador-20VE:~$
```

pstree

```
alumno@administrador-20VE:~$ pstree
systemd--ModemManager--3*[{ModemManager}]
systemd--NetworkManager--3*[{NetworkManager}]
systemd--accounts-daemon--3*[{accounts-daemon}]
systemd--avahi-daemon--avahi-daemon
systemd--bluetoothd
systemd--boltd--3*[{boltd}]
systemd--colord--3*[{colord}]
systemd--containerd--12*[{containerd}]
systemd--cron
systemd--cups-browsed--3*[{cups-browsed}]
systemd--cupsd
systemd--dbus-daemon
systemd--dockerd--13*[{dockerd}]
systemd--fwupd--5*[{fwupd}]
systemd--gdm3--gdm-session-wor--gdm-wayland-ses--gnome-session-b--3*[{gnom+
|                                     |                                     |3*[{gdm-wayland-ses}]
|                                     |3*[{gdm-session-wor}]
|3*[{gdm3}]
systemd--irqbalance--{irqbalance}
systemd--2*[{kerneloops}]
systemd--polkitd--3*[{polkitd}]
systemd--power-profiles--3*[{power-profiles-}]
systemd--rsyslogd--3*[{rsyslogd}]
systemd--rtkit-daemon--2*[{rtkit-daemon}]
systemd--snapd--14*[{snapd}]
systemd--switcheroo-cont--3*[{switcheroo-cont}]
systemd--(sd-pam)
systemd--at-spi2-registr--3*[{at-spi2-registr}]
systemd--dbus-daemon
systemd--dconf-service--3*[{dconf-service}]
systemd--evolution-addre--6*[{evolution-addre}]
systemd--evolution-calen--9*[{evolution-calen}]
systemd--evolution-sourc--4*[{evolution-sourc}]
systemd--gcr-ssh-agent--2*[{gcr-ssh-agent}]
systemd--2*[{gjs--11*[{gjs}]]
systemd--gnome-keyring-d--4*[{gnome-keyring-d}]
systemd--gnome-session-b--at-spi-bus-laun--dbus-daemon
|                                     |4*[{at-spi-bus-laun}]
|                                     |evolution-alarm--6*[{evolution-alarm}]
|                                     |gsd-disk-utilit--3*[{gsd-disk-utilit}]
|                                     |update-notifier--4*[{update-notifier}]
|                                     |4*[{gnome-session-b}]
systemd--gnome-session-c--{gnome-session-c}
systemd--gnome-shell--Xwayland--9*[{Xwayland}]
```

```
ps -u root
```

Esto mostrará todos los procesos del usuario "root".

```

alumno@administrador-20VE:~$ ps -u root
  PID TTY          TIME CMD
    1 ?        00:00:01 systemd
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 rcu_gp
    4 ?        00:00:00 rcu_par_gp
    5 ?        00:00:00 slub_flushwq
    6 ?        00:00:00 netns
    8 ?        00:00:00 kworker/0:0H-events_highpri
   10 ?        00:00:00 mm_percpu_wq
   11 ?        00:00:00 rcu_tasks_kthread
   12 ?        00:00:00 rcu_tasks_rude_kthread
   13 ?        00:00:00 rcu_tasks_trace_kthread
   14 ?        00:00:00 ksoftirqd/0
   15 ?        00:00:06 rcu_preempt
   16 ?        00:00:00 migration/0
   17 ?        00:00:00 idle_inject/0
   19 ?        00:00:00 cpuhp/0
   20 ?        00:00:00 cpuhp/1
   21 ?        00:00:00 idle_inject/1
   22 ?        00:00:00 migration/1
   23 ?        00:00:00 ksoftirqd/1
   25 ?        00:00:00 kworker/1:0H-events_highpri
   26 ?        00:00:00 cpuhp/2
   27 ?        00:00:00 idle_inject/2
   28 ?        00:00:00 migration/2
   29 ?        00:00:00 ksoftirqd/2
   31 ?        00:00:00 kworker/2:0H-events_highpri
   32 ?        00:00:00 cpuhp/3
   33 ?        00:00:00 idle_inject/3
   34 ?        00:00:00 migration/3
   35 ?        00:00:00 ksoftirqd/3
   37 ?        00:00:00 kworker/3:0H-events_highpri
   38 ?        00:00:00 cpuhp/4
   39 ?        00:00:00 idle_inject/4
   40 ?        00:00:00 migration/4
   41 ?        00:00:00 ksoftirqd/4
   43 ?        00:00:00 kworker/4:0H-events_highpri
   44 ?        00:00:00 cpuhp/5
   45 ?        00:00:00 idle_inject/5
   46 ?        00:00:00 migration/5
   47 ?        00:00:01 ksoftirqd/5
   49 ?        00:00:00 kworker/5:0H-events_highpri
   50 ?        00:00:00 cpuhp/6

```

5. Escribe un script para verificar y reiniciar automáticamente un proceso si no está corriendo.

```

GNU nano 7.2          verificar_reiniciar_proceso.sh
#!/bin/bash
if pgrep -f "verificar_reiniciar_proceso" > /dev/null
then
    echo "El proceso está corriendo"
else
    echo "El proceso no está corriendo, reiniciando..."
    ./verificar_reiniciar_proceso.sh &
fi

```

```

alumno@administrador-20VE:~$ nano verificar_reiniciar_proceso.sh
alumno@administrador-20VE:~$ chmod +x verificar_reiniciar_proceso.sh
alumno@administrador-20VE:~$ ./verificar_reiniciar_proceso.sh
El proceso está corriendo

```

En estos scripts (recuerda son archivos Bash, terminan en .sh) verifica que efectivamente hacen lo que se indica:

Monitoreo de procesos por uso excesivo de CPU

```
#!/bin/bash
```

```
ps -eo pid,ppid,%cpu,cmd --sort=-%cpu | head -10 | while read pid ppid cpu cmd; do
```

```
if (( $(echo "$cpu > 80.0" | bc -l) )); then
```

```
echo "Proceso $pid ($cmd) está utilizando $cpu% de CPU."
```

```
fi
```

```
done
```

```
GNU nano 7.2                                     monitoreo_cpu.sh
#!/bin/bash
ps -eo pid,ppid,%cpu,cmd --sort=-%cpu | head -11 | tail -n +2 | while read pid ppid cpu cmd; do
    echo "PID: $pid, PPID: $ppid, CPU: $cpu, CMD: $cmd"
    if (( $(echo "$cpu > 80.0" | bc -l) )); then
        echo "Proceso $pid ($cmd) está utilizando $cpu% de CPU."
    fi
done
```

```
alumno@administrador-20VE:~$ bash monitoreo_cpu.sh
PID: 8951, PPID: 8950, CPU: 100, CMD: ps -eo pid,ppid,%cpu,cmd --sort=-%cpu
Proceso 8951 (ps -eo pid,ppid,%cpu,cmd --sort=-%cpu) está utilizando 100% de CPU
.
PID: 5510, PPID: 3470, CPU: 23.7, CMD: /snap/firefox/4090/usr/lib/firefox/firefox
-contentproc -childID 17 -isForBrowser -prefsLen 31484 -prefMapSize 237467 -jsI
nitLen 234952 -parentBuildID 20240401184549 -greomni /snap/firefox/4090/usr/lib
/firefox/omni.js -appomni /snap/firefox/4090/usr/lib/firefox/browser/omni.js -app
Dir /snap/firefox/4090/usr/lib/firefox/browser {d09d5f2a-e458-42bf-bfea-567c19a
7cb50} 3470 true tab
PID: 3470, PPID: 2171, CPU: 23.0, CMD: /snap/firefox/4090/usr/lib/firefox/firefox
-contentproc -childID 17 -isForBrowser -prefsLen 31484 -prefMapSize 237467 -jsI
nitLen 234952 -parentBuildID 20240401184549 -greomni /snap/firefox/4090/usr/lib
/firefox/omni.js -appomni /snap/firefox/4090/usr/lib/firefox/browser/omni.js -app
Dir /snap/firefox/4090/usr/lib/firefox/browser {d09d5f2a-e458-42bf-bfea-567c19a
7cb50} 3470 true tab
PID: 8065, PPID: 3470, CPU: 7.3, CMD: /snap/firefox/4090/usr/lib/firefox/firefox
-contentproc -childID 47 -isForBrowser -prefsLen 31531 -prefMapSize 237467 -jsI
nitLen 234952 -parentBuildID 20240401184549 -greomni /snap/firefox/4090/usr/lib
/firefox/omni.js -appomni /snap/firefox/4090/usr/lib/firefox/browser/omni.js -app
Dir /snap/firefox/4090/usr/lib/firefox/browser {0fe88f32-b87a-446e-bcbb-a6375177
6506} 3470 true tab
PID: 2171, PPID: 1977, CPU: 6.7, CMD: /usr/bin/gnome-shell
PID: 8546, PPID: 3470, CPU: 5.5, CMD: /snap/firefox/4090/usr/lib/firefox/firefox
-contentproc -childID 52 -isForBrowser -prefsLen 31531 -prefMapSize 237467 -jsI
nitLen 234952 -parentBuildID 20240401184549 -greomni /snap/firefox/4090/usr/lib
/firefox/omni.js -appomni /snap/firefox/4090/usr/lib/firefox/browser/omni.js -app
Dir /snap/firefox/4090/usr/lib/firefox/browser {12f58a3d-3233-4a9c-8ab7-decfd990
a86e} 3470 true tab
PID: 7836, PPID: 3470, CPU: 2.1, CMD: /snap/firefox/4090/usr/lib/firefox/firefox
-contentproc -childID 45 -isForBrowser -prefsLen 31531 -prefMapSize 237467 -jsI
nitLen 234952 -parentBuildID 20240401184549 -greomni /snap/firefox/4090/usr/lib
/firefox/omni.js -appomni /snap/firefox/4090/usr/lib/firefox/browser/omni.js -app
Dir /snap/firefox/4090/usr/lib/firefox/browser {84f6cfe0-863c-4d0c-9d3a-4e564a8d
0929} 3470 true tab
PID: 4516, PPID: 3470, CPU: 1.3, CMD: /snap/firefox/4090/usr/lib/firefox/firefox
-contentproc -childID 11 -isForBrowser -prefsLen 31435 -prefMapSize 237467 -jsI
nitLen 234952 -parentBuildID 20240401184549 -greomni /snap/firefox/4090/usr/lib
/firefox/omni.js -appomni /snap/firefox/4090/usr/lib/firefox/browser/omni.js -app
Dir /snap/firefox/4090/usr/lib/firefox/browser {2e264718-52c5-4ac2-989f-475a1e90
be6f} 3470 true tab
PID: 8944, PPID: 1977, CPU: 0.7, CMD: /usr/libexec/tracker-extract-3
PID: 4044, PPID: 3470, CPU: 0.7, CMD: /snap/firefox/4090/usr/lib/firefox/firefox
-contentproc -childID 5 -isForBrowser -prefsLen 31297 -prefMapSize 237467 -jsIn
itLen 234952 -parentBuildID 20240401184549 -greomni /snap/firefox/4090/usr/lib/f
irefox/omni.js -appomni /snap/firefox/4090/usr/lib/firefox/browser/omni.js -appD
ir /snap/firefox/4090/usr/lib/firefox/browser {907766ab-1cbf-409c-8568-a2d60f6bd
b57} 3470 true tab
```

Identificar procesos zombis y reportar

```
#!/bin/bash
```

```
ps -eo stat,pid,cmd | grep "^Z" | while read stat pid cmd; do
```

```
    echo "Proceso zombi detectado: PID=$pid CMD=$cmd"
```

```
done
```



```
Archivo Editor Ver Buscar Terminal Ayuda
GNU nano 7.2 identificar_zombis.sh
#!/bin/bash
ps -eo stat,pid,cmd | grep "^Z" | while read stat pid cmd; do
    echo "Proceso zombi detectado: PID=$pid CMD=$cmd"
done
```

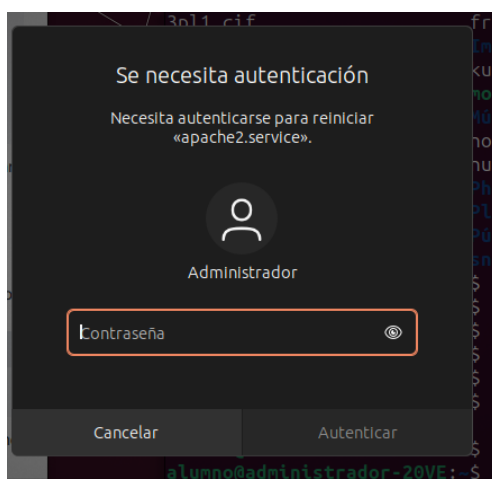
```
Archivo Editor Ver Buscar Terminal Ayuda
GNU nano 7.2 identificar_zombis.sh
#!/bin/bash
zombies=$(ps -eo stat,pid,cmd | awk '$1 ~ /Z/ {print "Proceso zombi detectado: PID=" $2 " CMD=" $3}')
if [ -n "$zombies" ]; then
    echo "$zombies"
else
    echo "No se han detectado procesos zombis."
fi
```

```
No se han detectado procesos zombis.
alumno@administrador-20VE:~$ nano identificar_zombis.sh
alumno@administrador-20VE:~$ bash identificar_zombis.sh
No se han detectado procesos zombis.
```

Reiniciar automáticamente un servicio no está corriendo

```
#!/bin/bash
SERVICE="apache2"
if ! ps -C $SERVICE > /dev/null; then
    systemctl restart $SERVICE
    echo "$SERVICE ha sido reiniciado."
fi
```

```
Archivo Editor Ver Buscar Terminal Ayuda
GNU nano 7.2 reiniciar_servicio.sh
#!/bin/bash
SERVICE="apache2"
if ! pgrep -x "$SERVICE" > /dev/null; then
    systemctl restart "$SERVICE"
    echo "$SERVICE ha sido reiniciado."
fi
```



```
alumno@administrador-20VE:~$ nano reiniciar_servicio.sh
alumno@administrador-20VE:~$ bash reiniciar_servicio.sh
Failed to restart apache2.service: Expiró el tiempo de conexión
See system logs and 'systemctl status apache2.service' for details.
apache2 ha sido reiniciado.
```

Verificar la cantidad de instancias de un proceso y actuar si supera un umbral

```
#!/bin/bash
PROCESS_NAME="httpd"
MAX_INSTANCES=10
count=$(ps -C $PROCESS_NAME --no-headers | wc -l)
if [ $count -gt $MAX_INSTANCES ]; then
    echo "Número máximo de instancias ($MAX_INSTANCES) superado para
$PROCESS_NAME con $count instancias."
fi
```

Puede pegar la Imagen desde el portapapeles.

```
GNU nano 7.2                                verificar_instancias.sh
#!/bin/bash
PROCESS_NAME="httpd"
MAX_INSTANCES=10
count=$(ps -C $PROCESS_NAME h | wc -l)
if [ $count -gt $MAX_INSTANCES ]; then
    echo "Número máximo de instancias ($MAX_INSTANCES) superado para $PROCESS_NAME con $count instancias."
else
    echo "El número de instancias de $PROCESS_NAME es $count, lo que está dentro del umbral establecido."
fi
```

```
alumno@administrador-20VE:~$ nano verificar_instancias.sh
alumno@administrador-20VE:~$ bash verificar_instancias.sh
El número de instancias de httpd es 0, lo que está dentro del umbral establecido.
```

Listar todos los procesos de usuarios sin privilegios (UID > 1000)

```
#!/bin/bash
ps -eo uid,pid,cmd | awk '$1 > 1000 {print}'
```

```
GNU nano 7.2                                listar_procesos_sin_privilegios.sh
#!/bin/bash
ps -eo uid,pid,cmd | awk '$1 > 1000 {print}'
```



```

alumno@administrador-20VE:~$ nano listar_procesos_sin_privilegios.sh
alumno@administrador-20VE:~$ bash listar_procesos_sin_privilegios.sh
  UID      PID  CMD
1001      1977 /lib/systemd/systemd --user
1001      1978 (sd-pam)
1001      1986 /usr/bin/pipewire
1001      1989 /usr/bin/ubuntu-report service
1001      1992 /usr/bin/wireplumber
1001      1999 /usr/bin/pipewire-pulse
1001      2003 /usr/bin/gnome-keyring-daemon --foreground --components=pkcs11,secrets --control-directory=/run/
user/1001/keyring
1001      2004 /usr/bin/dbus-daemon --session --address=systemd: --nofork --nopidfile --systemd-activation --sy
slog-only
1001      2031 /usr/libexec/xdg-document-portal
1001      2035 /usr/libexec/xdg-permission-store
1001      2066 /usr/libexec/gdm-wayland-session env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --se
ssion=ubuntu
1001      2070 /usr/libexec/gnome-session-binary --session=ubuntu
1001      2122 /usr/libexec/gcr-ssh-agent /run/user/1001/gcr
1001      2124 /usr/libexec/gnome-session-ctl --monitor
1001      2126 ssh-agent -D -a /run/user/1001/openssh_agent
1001      2136 /usr/libexec/gvfsd
1001      2142 /usr/libexec/gvfsd-fuse /run/user/1001/gvfs -f
1001      2144 /usr/libexec/gnome-session-binary --systemd-service --session=ubuntu
1001      2166 /usr/libexec/at-spi-bus-launcher --launch-immediately
1001      2171 /usr/bin/gnome-shell
1001      2178 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --pri
nt-address 11 --address=unix:path=/run/user/1001/at-spi/bus
1001      2260 /usr/libexec/gnome-shell-calendar-server
1001      2267 /usr/libexec/evolution-source-registry
1001      2280 /usr/libexec/goa-daemon
1001      2281 /usr/libexec/gvfs-udisks2-volume-monitor
1001      2298 /usr/libexec/evolution-calendar-factory
1001      2300 /usr/libexec/goa-identity-service
1001      2306 /usr/libexec/gvfs-goa-volume-monitor
1001      2315 /usr/libexec/gvfs-mtp-volume-monitor
1001      2320 /usr/libexec/gvfs-afc-volume-monitor
1001      2328 /usr/libexec/gvfs-gphoto2-volume-monitor
1001      2337 /usr/libexec/evolution-addressbook-factory
1001      2350 /usr/bin/gjs /usr/share/gnome-shell/org.gnome.Shell.Notifications
1001      2374 /usr/libexec/gvfsd-trash --spawner :1.18 /org/gtk/gvfs/exec_spaw/0
1001      2390 /usr/libexec/at-spi2-registryd --use-gnome-session
1001      2397 sh -c /usr/bin/ibus-daemon --panel disable ${[ "$XDG_SESSION_TYPE" = "x11" ] && echo "--xim")
1001      2398 /usr/libexec/gsd-a11y-settings
1001      2400 /usr/libexec/gsd-color
1001      2401 /usr/bin/ibus-daemon --panel disable

```

DE OTRA MANERA

Archivo Editar Ver Buscar Terminal Ayuda

GNU nano 7.2 listar_procesos_sin_privilegios.sh

#!/bin/bash

ps -eo user=,pid=,cmd= | awk -F: '\$1 != "root" {print \$0}'

```

alumno@administrador-20VE:~$ nano listar_procesos_sin_privilegios.sh
alumno@administrador-20VE:~$ bash listar_procesos_sin_privilegios.sh

```

```

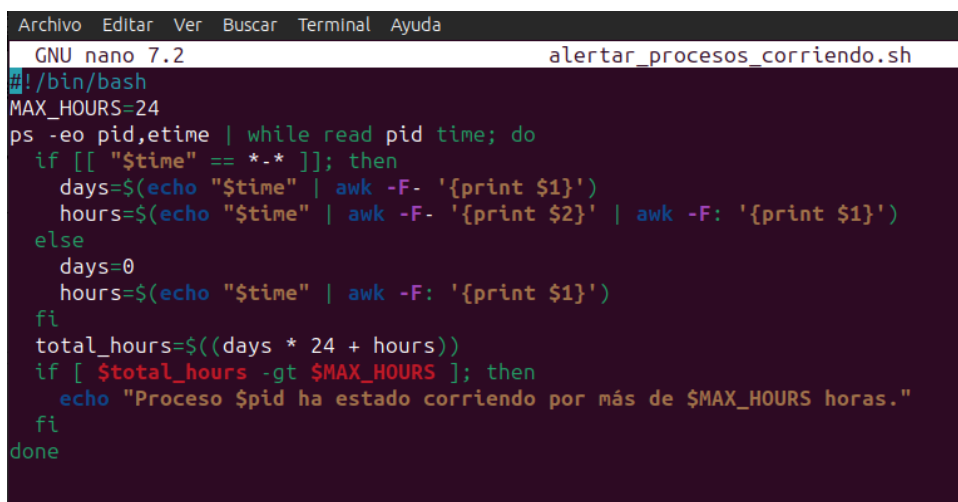
root      1 /sbin/init splash
root      2 [kthreadd]
root      3 [rcu_gp]
root      4 [rcu_par_gp]
root      5 [slub_flushwq]
root      6 [netns]
root      8 [kworker/0:0H-events_highpri]
root     10 [mm_percpu_wq]
root     11 [rcu_tasks_kthread]
root     12 [rcu_tasks_rude_kthread]
root     13 [rcu_tasks_trace_kthread]
root     14 [ksoftirqd/0]
root     15 [rcu_preempt]
root     16 [migration/0]
root     17 [idle_inject/0]
root     19 [cpuhp/0]
root     20 [cpuhp/1]
root     21 [idle_inject/1]
root     22 [migration/1]
root     23 [ksoftirqd/1]
root     25 [kworker/1:0H-events_highpri]
root     26 [cpuhp/2]
root     27 [idle_inject/2]
root     28 [migration/2]
root     29 [ksoftirqd/2]
root     31 [kworker/2:0H-events_highpri]
root     32 [cpuhp/3]
root     33 [idle_inject/3]
root     34 [migration/3]
root     35 [ksoftirqd/3]
root     37 [kworker/3:0H-events_highpri]
root     38 [cpuhp/4]
root     39 [idle_inject/4]
root     40 [migration/4]
root     41 [ksoftirqd/4]
root     43 [kworker/4:0H-events_highpri]
root     44 [cpuhp/5]
root     45 [idle_inject/5]
root     46 [migration/5]
root     47 [ksoftirqd/5]
root     49 [kworker/5:0H-events_highpri]
root     50 [cpuhp/6]

```

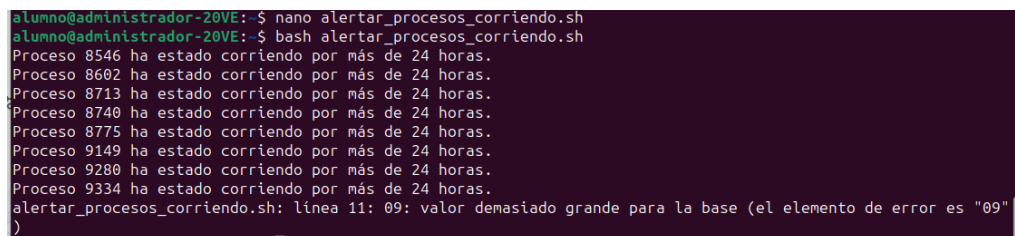
Alertar sobre procesos que han estado corriendo durante más de X horas

```
#!/bin/bash
MAX_HOURS=24
ps -eo pid,etime | while read pid time; do
    days=$(echo $time | grep -oP '^d+-' | sed 's:/-//')

    hours=$(echo $time | grep -oP '\d+:' | sed 's:/-//')
    total_hours=$((days * 24 + hours))
    if [ $total_hours -gt $MAX_HOURS ]; then
        echo "Proceso $pid ha estado corriendo por más de $MAX_HOURS horas."
    fi
done
```



```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
GNU nano 7.2                                alertar_procesos_corriendo.sh
#!/bin/bash
MAX_HOURS=24
ps -eo pid,etime | while read pid time; do
    if [[ "$time" == *.* ]]; then
        days=$(echo "$time" | awk -F- '{print $1}')
        hours=$(echo "$time" | awk -F- '{print $2}' | awk -F: '{print $1}')
    else
        days=0
        hours=$(echo "$time" | awk -F: '{print $1}')
    fi
    total_hours=$((days * 24 + hours))
    if [ $total_hours -gt $MAX_HOURS ]; then
        echo "Proceso $pid ha estado corriendo por más de $MAX_HOURS horas."
    fi
done
```



```
alumno@administrador-20VE:~$ nano alertar_procesos_corriendo.sh
alumno@administrador-20VE:~$ bash alertar_procesos_corriendo.sh
Proceso 8546 ha estado corriendo por más de 24 horas.
Proceso 8602 ha estado corriendo por más de 24 horas.
Proceso 8713 ha estado corriendo por más de 24 horas.
Proceso 8740 ha estado corriendo por más de 24 horas.
Proceso 8775 ha estado corriendo por más de 24 horas.
Proceso 9149 ha estado corriendo por más de 24 horas.
Proceso 9280 ha estado corriendo por más de 24 horas.
Proceso 9334 ha estado corriendo por más de 24 horas.
alertar_procesos_corriendo.sh: línea 11: 09: valor demasiado grande para la base (el elemento de error es "09")
)
```

Encontrar y listar todos los procesos que escuchan en un puerto específico

```
#!/bin/bash
PORT="80"
lsof -i :$PORT | awk 'NR > 1 {print $2}' | while read pid; do
    ps -p $pid -o pid,cmd
done
```

```

GNU nano 7.2 encontrar_procesos_en_puerto.sh
#!/bin/bash
PORT="80"
echo "Procesos escuchando en el puerto $PORT:"
lsof -i :$PORT | awk 'NR > 1 {print $2}' | while read pid; do
  ps -p $pid -o pid,cmd --no-headers
done

```

```

alumno@administrador-20VE:~$ nano encontrar_procesos_en_puerto.sh
alumno@administrador-20VE:~$ bash encontrar_procesos_en_puerto.sh
Procesos escuchando en el puerto 80:

```

Monitorear la memoria utilizada por un conjunto de procesos y alertar si supera un umbral

```

#!/bin/bash
PROCESS_NAME="mysqld"
MAX_MEM=1024 # 1GB en MB
ps -C $PROCESS_NAME -o pid,rss | while read pid rss; do
  if [ $rss -gt $MAX_MEM ]; then
    echo "Proceso $pid ($PROCESS_NAME) está utilizando más de $MAX_MEM MB de memoria."
  fi
done

```

```

GNU nano 7.2 monitorear_memoria.sh
#!/bin/bash

# Nombre del proceso a monitorear
PROCESS_NAME="mysqld"

# Umbral de memoria en MB (1GB)
MAX_MEM=1024 # 1GB en MB

# Usar ps para obtener PID y RSS de los procesos que coinciden con el nombre
ps -C "$PROCESS_NAME" -o pid,rss= --no-headers | while read -r pid rss; do
  # Convertir el uso de memoria de KB a MB
  rss_mb=$((rss / 1024))

  # Verificar si el uso de memoria supera el umbral
  if [[ $rss_mb -gt $MAX_MEM ]]; then
    echo "Proceso $pid ($PROCESS_NAME) está utilizando más de $MAX_MEM MB de memoria."
  fi
done

```

```

alumno@administrador-20VE:~$ nano monitorear_memoria.sh
alumno@administrador-20VE:~$ bash monitorear_memoria.sh
alumno@administrador-20VE:~$ ps aux | grep mysqld
alumno      12173  0.0  0.0  6576  2432 pts/0    S+   13:41   0:00 grep --color=auto mysqld

```

Generar un informe de procesos que incluya PID, tiempo de ejecución y comando

```

#!/bin/bash
ps -eo pid,etime,cmd --sort=-etime | head -20 > proceso_informe.txt
echo "Informe generado en proceso_informe.txt."

```

```
GNU nano 7.2                               informe_procesos.sh
#!/bin/bash
echo -e "PID\tTiempo de Ejecución\tComando" > proceso_informe.txt
ps -eo pid,etime,cmd --sort=-etime | head -n 20 | awk '{print $1 "\t" $2 "\t" $3}' >> proceso_informe.txt
echo "Informe generado en proceso_informe.txt."
```

```
alumno@administrador-20VE:~$ nano informe_procesos.sh
alumno@administrador-20VE:~$ bash informe_procesos.sh
Informe generado en proceso_informe.txt.
```

El comando grep

El comando **grep** es una herramienta de línea de comandos disponible en sistemas Unix y Linux utilizada para buscar texto dentro de archivos o flujos de datos. El nombre **grep** proviene de "global regular expression print", refiriéndose a su capacidad para filtrar líneas de texto que coinciden con expresiones regulares especificadas.

grep es extremadamente útil para analizar archivos de log, buscar ocurrencias de cadenas de texto en archivos de código, filtrar output de otros comandos, y muchas otras tareas de búsqueda de texto.

En el contexto de la computación paralela, concurrente y distributiva, así como en la automatización, **grep** se puede aplicar de diversas formas:

- **Análisis de logs de aplicaciones distribuidas:** **grep** puede ser utilizado para buscar rápidamente mensajes de error, advertencias o eventos específicos dentro de grandes volúmenes de archivos de log generados por aplicaciones distribuidas, facilitando el diagnóstico de problemas.
- **Monitoreo de salud del sistema:** Al integrarse en scripts de shell, **grep** puede automatizar el monitoreo del estado de servicios y procesos críticos, extrayendo información relevante de comandos como **ps**, **netstat**, o archivos como **/proc/meminfo**.
- **Validación de configuraciones en clusters:** **grep** puede ser utilizado para verificar rápidamente la consistencia de configuraciones de software en nodos de un cluster, buscando discrepancias o configuraciones erróneas en archivos distribuidos.
- **Automatización de tareas de gestión:** Integrado en scripts de shell, **grep** puede automatizar la gestión de recursos computacionales, por ejemplo, identificando y respondiendo a condiciones específicas detectadas en logs o salidas de comandos.
- **Análisis de rendimiento y carga de trabajo:** **grep** es útil para filtrar datos específicos de rendimiento y carga de trabajo de herramientas de monitoreo y métricas, permitiendo a los desarrolladores y administradores centrarse en información relevante para la optimización.

Ejercicios

En estos scripts (recuerda son archivos Bash, terminan en .sh) verifica que efectivamente hacen lo que se indica:

Filtrar errores específicos en logs de aplicaciones paralelas:

grep "ERROR" /var/log/myapp/*.log

```
alumno@administrador-20VE:~$ sudo grep "ERROR" /var/log/nginx/*.log
alumno@administrador-20VE:~$
```

Verificar la presencia de un proceso en múltiples nodos:

pdsh -w nodo[1-10] "ps aux | grep 'my_process' | grep -v grep"

```
alumno@administrador-20VE:~$ pdsh -w nodo[1-10] "ps aux | grep 'my_process' | grep -v grep"
pdsh@administrador-20VE: gethostname("nodo1") failed
alumno@administrador-20VE:~$
```

Contar el número de ocurrencias de condiciones de carrera registradas:

grep -c "race condition" /var/log/myapp.log

```
alumno@administrador-20VE:~$ grep -c "race condition" app-web
grep: app-web: Es un directorio
0
alumno@administrador-20VE:~$
```

Extraer IPs que han accedido concurrentemente a un recurso:

grep "accessed resource" /var/log/webserver.log | awk '{print \$1}' | sort | uniq

```
alumno@administrador-20VE:~$ sudo grep "accessed resource" /var/log/auth.log | a
wk '{print $1}' | sort | uniq
2024-04-15T14:24:57.244585-05:00
alumno@administrador-20VE:~$
```

Automatizar la alerta de sobrecarga en un servicio distribuido:

grep "out of memory" /var/log/services/*.log && mail -s "Alerta de Memoria" admin@example.com < /dev/null

```
alumno@administrador-20VE:~$ sudo grep "out of memory" /var/log/apt/*.log && mai
l -s "Alerta de Memoria" alumno@administrador-20VE < /dev/null
```

Monitorear errores de conexión en aplicaciones concurrentes:

grep -i "connection error" /var/log/myapp_error.log | mail -s "Errores de Conexión Detectados" admin@example.com

Validar la correcta sincronización en operaciones distribuidas:

```
grep "operation completed" /var/logs/distributed_app/*.log | awk '{print $5, $NF}' |
```

```
sort
```

```
alumno@administrador-20VE:~$ sudo grep "operation completed" /var/log/apt/*.log  
| awk '{print $5, $NF}' | sort
```

Monitorizar la creación de procesos no autorizados

```
#!/bin/bash
```

```
watch -n 5 'ps aux | grep -vE "(root|daemon|nobody)" | grep -v grep'
```

```
alumno@administrador-20VE:~$ nano creacio.sh  
alumno@administrador-20VE:~$ bash creacio.sh
```

```
Cada 5,0s: ps aux | grep -vE "(root|daemon|nobody)" | gre... administrador-20VE: Mon Apr 3
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
systemd+	560	0.1	0.0	16204	7168	?	Ss	12:43	0:07	/lib/systemd/systemd-oom
systemd+	561	0.0	0.0	20488	12416	?	Ss	12:43	0:02	/lib/systemd/systemd-res
systemd+	562	0.0	0.0	89692	7296	?	Ssl	12:43	0:00	/lib/systemd/systemd-ti
www-data	810	0.0	0.0	58400	5540	?	S	12:43	0:00	nginx: worker process
www-data	811	0.0	0.0	58400	5540	?	S	12:43	0:00	nginx: worker process
www-data	812	0.0	0.0	58400	5540	?	S	12:43	0:00	nginx: worker process
www-data	813	0.0	0.0	58400	5284	?	S	12:43	0:00	nginx: worker process
www-data	814	0.0	0.0	58400	5540	?	S	12:43	0:00	nginx: worker process
www-data	815	0.0	0.0	58400	5540	?	S	12:43	0:00	nginx: worker process
www-data	816	0.0	0.0	58400	5540	?	S	12:43	0:00	nginx: worker process
www-data	817	0.0	0.0	58400	5540	?	S	12:43	0:00	nginx: worker process
nvidia+	841	0.0	0.0	5152	2048	?	Ss	12:43	0:00	/usr/bin/nvidia-persiste
colord	1368	0.0	0.0	316520	12924	?	Ssl	12:43	0:00	/usr/libexec/colord
cups-br+	1693	0.0	0.1	260424	17280	?	Ssl	12:43	0:00	/usr/sbin/cups-browsed
kernoops	1751	0.0	0.0	12520	2056	?	Ss	12:43	0:00	/usr/sbin/kerneloops --t
kernoops	1756	0.0	0.0	12520	2064	?	Ss	12:43	0:00	/usr/sbin/kerneloops
alumno	2100	0.0	0.0	19740	11520	?	Ss	12:44	0:00	/lib/systemd/systemd --t
alumno	2101	0.0	0.0	170468	6788	?	S	12:44	0:00	(sd-pam)
alumno	2108	0.0	0.0	65500	15972	?	S<sl	12:44	0:01	/usr/bin/pipewire
alumno	2111	0.0	0.0	1153560	8832	?	Ssl	12:44	0:00	/usr/bin/ubuntu-report s
alumno	2112	0.0	0.1	329576	17408	?	S<sl	12:44	0:01	/usr/bin/wireplumber
alumno	2124	0.0	0.1	46976	20088	?	S<sl	12:44	0:00	/usr/bin/pipewire-pulse
alumno	2153	0.0	0.0	534128	7680	?	Ssl	12:44	0:00	/usr/libexec/xdg-documen
alumno	2157	0.0	0.0	307088	6144	?	Ssl	12:44	0:00	/usr/libexec/xdg-permiss
alumno	2233	0.0	0.0	233252	6144	tty2	Ssl+	12:44	0:00	/usr/libexec/gdm-x-sessi
alumno	2235	2.1	0.9	26897084	151840	tty2	Sl+	12:44	2:23	/usr/lib/xorg/Xorg vt2

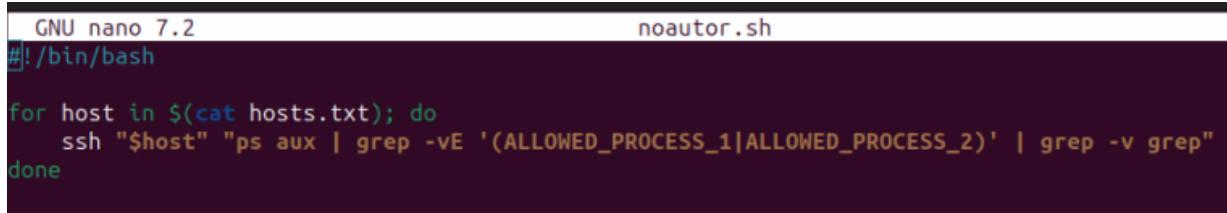
Detectar y alertar sobre ataques de fuerza bruta SSH

```
grep "Failed password" /var/log/auth.log | cut -d' ' -f11 | sort | uniq -c | sort -nr | head
```

```
alumno@administrador-20VE:~$ sudo grep "Failed password" /var/log/auth.log  
| cut -d' ' -f11 | sort | uniq -c | sort -nr | head  
1 ;
```

Identificar uso no autorizado de recursos en clústeres de computación

```
#!/bin/bash
for host in $(cat hosts.txt); do
  ssh $host "ps aux | grep -vE '(ALLOWED_PROCESS_1|ALLOWED_PROCESS_2)' | grep -v grep"
done
```

A screenshot of a terminal window with a dark purple background. The title bar at the top shows "GNU nano 7.2" on the left and "noautor.sh" on the right. The terminal text is as follows:

```
#!/bin/bash
for host in $(cat hosts.txt); do
  ssh "$host" "ps aux | grep -vE '(ALLOWED_PROCESS_1|ALLOWED_PROCESS_2)' | grep -v grep"
done
```

Pipes

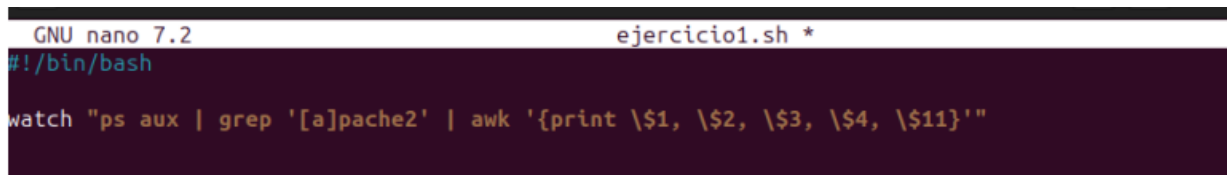
Un "pipe" en Linux, simbolizado por |, es una poderosa característica de la línea de comandos que permite pasar la salida (output) de un comando directamente como entrada (input) a otro comando. Esto facilita la creación de secuencias de comandos o pipelines donde el resultado de un proceso es inmediatamente utilizado por otro, permitiendo una manipulación de datos eficiente y flexible sin necesidad de archivos intermedios.

En el contexto de la computación paralela, concurrente y distributiva, los pipes son fundamentales para procesar y analizar datos generados por múltiples procesos, monitorizar el rendimiento y estado de sistemas distribuidos y automatizar tareas administrativas complejas. Al combinar **ps**, **grep** y otros comandos de Linux, se pueden crear pipelines eficientes para la gestión y análisis de sistemas.

Ejercicios

Indica las actividades que realizan cada uno de los scripts (recuerda son archivos Bash y por tanto terminan en .sh y cada línea representa un script diferente)

```
watch "ps aux | grep '[a]pache2' | awk '{print \$1, \$2, \$3, \$4, \$11}'"
```

A screenshot of a terminal window with a dark purple background. The title bar at the top shows "GNU nano 7.2" on the left and "ejercicio1.sh *" on the right. The terminal text is as follows:

```
#!/bin/bash
watch "ps aux | grep '[a]pache2' | awk '{print \$1, \$2, \$3, \$4, \$11}'"
```



```
cat /var/log/myapp.log | grep "ERROR" | awk '{print $NF}' | sort | uniq -c | sort -nr
```

```
GNU nano 7.2                                ejercicio2.sh *
#!/bin/bash
cat /var/log/myapp.log | grep "ERROR" | awk '{print $NF}' | sort | uniq -c | sort -nr
```

```
alumno@administrador-20VE:~$ sudo bash ejercicio2.sh
```

```
systemctl --failed | grep "loaded units listed" || systemctl restart $(awk '{print
```

```
$1}')
```

```
GNU nano 7.2                                ejercicio3.sh
#!/bin/bash
systemctl --failed | grep "loaded units listed" || systemctl restart $(awk '{print $1}')
```

```
alumno@administrador-20VE:~$ bash ejercicio3.sh
1 loaded units listed.
```

```
ps -eo pid,ppid,%cpu,cmd --sort=-%cpu | awk '$3 > 80 {print "Alto uso de CPU: ", $1}' | mail -s "Alerta CPU" admin@example.com
```

Con este ejemplo identificamos los procesos que excedan el 80% de uso de la CPU en el sistema y notificar al administrador mediante correo electrónico.

```
ls /var/log/*.log | xargs -n 1 -P 5 -I {} ssh nodo_remoto "grep 'ERROR' {} > errores_{}.txt"
```

Este ejemplo explora todos los archivos con extensión ".log" en busca de la palabra "ERROR" y almacena los hallazgos en archivos individuales en el servidor remoto.

```
echo "8.8.8.8 www.example.com" | xargs -n 1 ping -c 1 | grep "bytes from" || echo "$(date)
Fallo de ping" >> fallos_ping.txt
```

```
alumno@administrador-20VE:~$ echo "8.8.8.8 www.example.com" | xargs -n 1 ping -c 1 | grep "bytes from" || echo "$(date)
Fallo de ping" >> fallos_ping.txt
64 bytes from 8.8.8.8: icmp_seq=1 ttl=54 time=36.2 ms
64 bytes from 93.184.216.34 (93.184.216.34): icmp_seq=1 ttl=52 time=80.5 ms
```

```
ps -eo user,%cpu,%mem,cmd | awk '/httpd/ {cpu+=$2; mem+=$3; count++} END {print "Apache - CPU:", cpu/count, "Mem:", mem/count}'
```

```
alumno@administrador-20VE:~$ ps -eo user,%cpu,%mem,cmd | awk '/httpd/ {cpu+=$2; mem+=$3; count++} END {print "Apache - CPU:", cpu/count, "Mem:", mem/count}'
Apache - CPU: 0 Mem: 0
```

```
df /home | awk '$5 > 80 {print $1}' | xargs -l {} tar -czf "{}_$(date +%F).tar.gz" {}
```

```
import subprocess
```

```
# Ejecutar el comando ps y obtener la salida
```

```
result = subprocess.run(['ps', '-eo', '%cpu,pid,cmd'],
```

```
stdout=subprocess.PIPE) lines = result.stdout.decode('utf-8').strip().split('\n')
```

```
# Analizar cada línea de la salida de ps
```

```
for line in lines[1:]: # Saltar la primera línea que es la cabecera
```

```
    cpu_usage, pid, cmd = line.split(None, 2)
```

```
    if float(cpu_usage) > 80.0: # Umbral de uso de CPU
```

```
        print(f"Alerta: Proceso {pid} ({cmd}) está utilizando {cpu_usage}% de CPU")
```

- **Muestra información sobre los procesos en un sistema Unix. Para**

cada línea de salida (excepto la primera, que es la cabecera), verifica

si el uso de CPU supera el 80%. Si es así, emite una alerta indicando

el ID del proceso, el comando y el uso de CPU.

.....

```
import subprocess
```

```
# Filtrar líneas con errores de un archivo de log
```

```
cmd = "grep 'ERROR' /var/log/myapp.log"
```

```
errors = subprocess.check_output(cmd, shell=True).decode('utf-8').split("\n")
```

- **Busca todas las líneas que contienen la palabra "ERROR" en el archivo de registro /var/log/myapp.log. Luego, lee la salida de este comando utilizando subprocess.check_output y la decodifica de UTF-8 a una cadena de texto. Finalmente, divide esta cadena en líneas individuales y almacena esas líneas en una lista llamada errors, que contendrá todas las líneas del archivo de registro que contienen la palabra "ERROR".**

```
# Analizar errores y contar ocurrencias
```

```
error_counts = {}
```

```
for error in errors:
```

```
    if error in error_counts:
```

```
        error_counts[error] += 1
```

```
    else:
```

```
        error_counts[error] = 1
```

- **crea un diccionario llamado error_counts para contar las ocurrencias de cada tipo de error encontrado en la lista errors. Recorre la lista errors y, para cada error, verifica si ya existe una entrada en el diccionario error_counts. Si el error ya está en el diccionario, incrementa su contador. Si no está en el diccionario, agrega una nueva entrada con un contador inicializado en 1. Al finalizar, el diccionario error_counts contendrá el recuento de cada tipo de error encontrado en la lista errors**

```
# Imprimir el recuento de errores
```

```
for error, count in error_counts.items():
```

```
    print(f"{error}: {count}")
```

- **muestra cómo imprimir el recuento de errores almacenados en el diccionario error_counts. Utiliza un bucle for para iterar sobre los elementos del diccionario, donde error es la clave (el tipo de error) y count es el valor (el recuento de ese**

tipo de error). Luego, utiliza print para mostrar cada tipo de error junto con su recuento.

```
.....
```

```
from multiprocessing import Pool

import subprocess

def analyze_log(file_part):

    """Función para analizar una parte del archivo de log."""

    with open(file_part) as f:

        return f.read().count('ERROR')
```

- **Define una función llamada analyze_log que toma una parte del archivo de registro (file_part) y cuenta las ocurrencias de la cadena 'ERROR' en esa parte del archivo.**

```
# Dividir el archivo de log en partes

subprocess.run(['split', '-l', '1000', 'large_log.log', 'log_part_'])
```

- **Utiliza el comando split de Unix para dividir el archivo large_log.log en partes más pequeñas, cada una con un máximo de 1000 líneas. Las partes resultantes se guardarán con el prefijo log_part_.**

```
# Lista de archivos divididos

parts = subprocess.check_output('ls log_part_*', shell=True).decode().split()
```

- **Utiliza el comando ls para obtener una lista de archivos que comienzan con log_part_ (los archivos divididos previamente). Luego, utiliza subprocess.check_output para ejecutar este comando, decodifica la salida en una cadena de texto y la divide en una lista, asignándole a la variable parts.**

```
# Utilizar multiprocessing para analizar las partes en paralelo

with Pool(4) as p:
```

```
results = p.map(analyze_log, parts)
```

```
print("Total de errores encontrados:", sum(results))
```

- **Utiliza la biblioteca multiprocessing para analizar las partes del archivo de registro en paralelo. Con Pool(4) se crea un grupo de procesos, en este caso con 4 procesos. Luego, p.map(analyze_log, parts) aplica la función analyze_log a cada elemento de la lista parts utilizando los procesos del grupo, lo que permite procesar varias partes del archivo de registro simultáneamente. Finalmente, suma los resultados de todas las partes para obtener el total de errores encontrados.**

```
.....  
. import subprocess
```

```
import time
```

```
previous_ports = set()
```

```
while True:
```

- **Importando los módulos subprocess y time. Luego, inicializa un conjunto vacío llamado previous_ports. A continuación, entra en un bucle infinito (while True:), lo que significa que seguirá ejecutándose continuamente hasta que se detenga manualmente.**

```
# Ejecutar netstat y capturar la salida
```

```
result = subprocess.run(['netstat', '-tuln'], stdout=subprocess.PIPE) ports =
```

```
set(line.split()[3] for line in result.stdout.decode().split("\n") if 'LISTEN' in line)
```

- **Utiliza subprocess.run para ejecutar el comando netstat -tuln, que muestra una lista de puertos en escucha en el sistema. La salida de este comando se captura y procesa para extraer los números de puerto que**

están en estado de "LISTEN". Estos números de puerto se almacenan en un conjunto llamado ports

```
# Detectar cambios en puertos abiertos

new_ports = ports - previous_ports

closed_ports = previous_ports - ports

if new_ports or closed_ports:

    print(f"Nuevos puertos abiertos: {new_ports}, Puertos cerrados: {closed_ports}")

    previous_ports = ports

time.sleep(60) # Esperar un minuto antes de volver a verificar

.....

. import subprocess
```

- **Compara el conjunto de puertos actuales (ports) con el conjunto de puertos anteriores (previous_ports) para detectar cambios en los puertos abiertos. Si hay nuevos puertos abiertos o puertos cerrados en comparación con la última verificación, se imprime un mensaje**

indicando los cambios. Luego, actualiza el conjunto previous_ports

con los puertos actuales y espera 60 segundos antes de realizar la

próxima verificación.

```
# Obtener uso de memoria por proceso
```

```
result = subprocess.run(['ps', '-eo', 'user,rss'], stdout=subprocess.PIPE)
```

```
lines = result.stdout.decode().split('\n')
```

- **Ejecuta el comando `ps -eo user,rss`, que muestra el nombre de usuario y el uso de memoria residente (RSS) de cada proceso en el sistema. La salida de este comando se captura y se divide en líneas, donde cada línea contiene información sobre un proceso.**

```
# Calcular el uso total de memoria por usuario
```

```
memory_usage = {}
```

```
for line in lines[1:-1]: # Ignorar la primera y última línea (cabecera y línea vacía)
```

```
user, rss = line.split(None, 1)
```

```
memory_usage[user] = memory_usage.get(user, 0) + int(rss)
```

```
for user, rss in memory_usage.items():
```

```
print(f"Usuario: {user}, Memoria RSS total: {rss} KB")
```

- **Calcula el uso total de memoria por usuario utilizando la salida del comando `ps`. Crea un diccionario `memory_usage` para almacenar el uso de memoria por**

usuario. Luego, itera sobre cada línea de la salida del comando (excepto la primera y la última) y extrae el nombre de usuario y la cantidad de memoria RSS utilizada por el proceso. El código suma la cantidad de memoria utilizada por cada proceso de un mismo usuario y almacena el resultado en el diccionario `memory_usage`. Finalmente, imprime el uso total de memoria RSS por cada usuario en el sistema.

```
.....

import subprocess

import datetime

snapshot_interval = 60 # en segundos

while True:

    timestamp = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')

    cpu_usage = subprocess.check_output("top -b -n1 | awk '/Cpu(s):/ {print $2}'",
    shell=True).decode().strip()

    memory_usage = subprocess.check_output("free | grep Mem | awk '{print $3/$2 * 100.0}'",
    shell=True).decode().strip()

    with open("system_performance.log", "a") as log_file:

        log_file.write(f'{timestamp}, CPU: {cpu_usage}%, Memoria: {memory_usage}%\n')

    time.sleep(snapshot_interval)

    - Crea un bucle infinito que captura el uso de CPU y memoria en intervalos regulares y los guarda en un archivo de registro system_performance.log. En cada iteración del bucle, se obtiene la marca de tiempo actual y se utiliza subprocess.check_output para ejecutar comandos que recuperan el uso de CPU y memoria. Estos valores se escriben en el archivo de registro junto con la marca de tiempo. El bucle espera entonces el intervalo especificado antes de tomar otra instantánea del sistema.

.....
```

```
. #!/bin/bash
```

```
while true; do
```

```
ps -eo %cpu,pid,cmd --sort=-%cpu | head -n 10 | awk '$1 > 80.0 {
```

```
printf("Alto uso de CPU (%s%%) por PID %s: %s\n", $1, $2, $3); }'
```

```
| while read LINE; do
```

```
echo "$LINE" | mail -s "Alerta de CPU" admin@domain.com
```

```
done
```

```
sleep 60
```

```
done
```

- **Este script verifica regularmente los procesos con mayor uso de CPU. Si encuentra algún proceso que esté utilizando más del 80% de la CPU, envía una alerta por correo electrónico al administrador. Luego, espera 60 segundos y repite el proceso.**

```
.....
```

```
#!/bin/bash
```

```
while true; do
```

```
ps -eo %cpu,pid,cmd --sort=-%cpu | head -n 10 | awk '$1 > 80.0 {
```

```
printf("Alto uso de CPU (%s%%) por PID %s: %s\n", $1, $2, $3); }'
```

```
| while read LINE; do
```

```
echo "$LINE" | mail -s "Alerta de CPU" admin@domain.com
```

```
done
```

```
sleep 60
```

```
done
```

- **Este script monitorea constantemente los procesos con mayor uso de CPU. Cada minuto, verifica los 10 procesos con mayor uso de CPU y, si alguno de ellos supera el 80%, envía una alerta por correo electrónico al administrador. Luego, el script espera otros 60 segundos antes de volver a realizar la verificación.**

```
#!/bin/bash
```

```
ps -eo user,rss | awk '{arr[$1]+=$2} END {  
  
for (user in arr) {  
  
print user, arr[user] " KB";  
  
}  
  
}' | sort -nrk 2 > /tmp/memory_usage_by_user.txt  
  
echo "Uso de memoria por usuario guardado en  
  
/tmp/memory_usage_by_user.txt."
```

- **Este script calcula el uso de memoria por usuario y guarda los resultados en un archivo de texto. Utiliza el comando ps para obtener el nombre de usuario y la memoria residente (RSS) de cada proceso, luego utiliza awk para sumar la memoria de todos los procesos de cada usuario. Finalmente, ordena los resultados por el uso de memoria y los guarda en el archivo /tmp/memory_usage_by_user.txt, mostrando un mensaje de confirmación.**

```
#!/bin/bash
```

```
echo "Top CPU y Memoria por Usuario"
```

```
ps -eo user,%cpu,%mem --sort=-%cpu | awk 'NR==1 {print $0; next} !seen[$1]++' | while  
read USER CPU MEM; do
```

```
echo "Usuario: $USER, CPU: $CPU%, Mem: $MEM%"
```

```
done
```

- Este script de muestra los principales usuarios en términos de uso de CPU y memoria. Utiliza el comando ps para obtener una lista de procesos ordenados por uso de CPU, luego utiliza awk para mostrar solo la primera línea (encabezado) y la primera instancia de cada usuario. Finalmente, muestra el nombre de usuario, el uso de CPU y el uso de memoria para cada usuario encontrado.

```
.....  
  
#!/bin/bash
```

```
PROCESS_NAME="java"
```

```
echo "Reporte de Memoria para procesos $PROCESS_NAME"
```

```
ps -C $PROCESS_NAME -o pid,user,%mem,cmd --sort=-%mem | awk 'NR==1; NR>1 {print $0; total+=$3} END {print "Memoria Total Usada:", total "%"}'
```

- Este script de genera un informe de uso de memoria para procesos con el nombre "java". Utiliza el comando ps para obtener una lista de procesos java ordenados por uso de memoria. Luego, utiliza awk para mostrar la primera línea (encabezado) y todas las líneas de los procesos java, sumando el uso de memoria de todos los procesos para mostrar la memoria total utilizada por estos procesos al final del informe

```
.....  
  
#!/bin/bash
```

```
LOG="/var/log/httpd/access_log"
```

```
echo "Top IPs"
```

```
awk '{print $1}' $LOG | sort | uniq -c | sort -nr | head -5 | while read COUNT IP; do
```

```
LOCATION=$(geoplookup $IP | cut -d, -f2)
```

```
echo "$IP ($COUNT accesos) - $LOCATION"
```

```
done
```

- Este script de muestra las cinco direcciones IP más frecuentes que acceden al archivo de registro de acceso de HTTP (access_log). Utiliza awk para extraer las

direcciones IP del archivo de registro, luego las ordena y cuenta las apariciones de cada una con uniq -c. Después, las ordena nuevamente por cantidad en orden descendente con sort -nr y muestra solo las cinco primeras con head -5. Para cada una de estas direcciones IP, utiliza geoiplookup para obtener la ubicación geográfica y muestra el resultado junto con el número de accesos.

.....

```
#!/bin/bash
```

```
NET_DEV="eth0"
```

```
echo "Estadísticas de red para $NET_DEV"
```

```
rx_prev=$(cat /sys/class/net/$NET_DEV/statistics/rx_bytes)
```

```
tx_prev=$(cat /sys/class/net/$NET_DEV/statistics/tx_bytes)
```

```
sleep 5
```

```
rx_now=$(cat /sys/class/net/$NET_DEV/statistics/rx_bytes)
```

```
tx_now=$(cat /sys/class/net/$NET_DEV/statistics/tx_bytes)
```

```
rx_rate=$(( ( $rx_now - $rx_prev ) / 5 ))
```

```
tx_rate=$(( ( $tx_now - $tx_prev ) / 5 ))
```

```
echo "RX Rate: $rx_rate bytes/sec"
```

```
echo "TX Rate: $tx_rate bytes/sec"
```

- **Este script de muestra las estadísticas de red para el dispositivo especificado (eth0 en este caso). Calcula las tasas de recepción y transmisión dividiendo la diferencia en bytes entre dos momentos diferentes (tomados con un intervalo de 5 segundos) por 5 para obtener la tasa por segundo. Luego, muestra estas tasas en bytes por segundo.**

Bash

Para profundizar en el aprendizaje y comprensión de Bash en el contexto de computación paralela, concurrente y distribuida, necesitarán una base sólida en varios conceptos y herramientas de línea de comandos. A continuación, les presento una lista de referencias y recursos que pueden ser útiles permitiéndoles no solo entender los scripts proporcionados aquí, sino también desarrollar sus propios scripts para resolver problemas complejos en estos entornos.

"The Linux Command Line" por William Shotts <https://linuxcommand.org/tlcl.php>

La documentación oficial de Bash es un recurso indispensable para comprender todas las características y capacidades del shell. <https://www.gnu.org/software/bash/manual/>

Explainshell : <https://explainshell.com/>

Un sitio web que desglosa comandos de Bash, mostrando una explicación detallada de cada parte de un comando.

Ryan's Tutorials - Bash Scripting Tutorial <https://ryanstutorials.net/bash-scripting-tutorial/>

Una herramienta de linting para scripts de Bash que ayuda a encontrar errores y problemas comunes en el código. ShellCheck: <https://www.shellcheck.net/>

Una herramienta para ejecutar tareas en paralelo utilizando la línea de comandos. Es muy útil para procesamiento de datos y tareas computacionales que pueden ser paralelizadas. GNU Parallel <https://www.gnu.org/software/parallel/>

Para escribir scripts efectivos en Bash, especialmente en el contexto de computación paralela, concurrente y distribuida, es esencial dominar ciertos fundamentos y conceptos avanzados de Bash. A continuación, presento un resumen de los aspectos clave de Bash que debes conocer, acompañados de ejemplos ilustrativos.

Variables: Almacenar y manipular datos.

```
nombre="Mundo"
```

```
echo "Hola, $nombre"
```

Estructuras de Control: Permiten tomar decisiones y repetir

acciones. # If statement

```
if [ "$nombre" == "Mundo" ]; then
```

```
    echo "Correcto"
```

```
fi
```

Loop

```
for i in {1..5}; do
```

```
    echo "Iteración $i"
```

```
done
```


Funciones: Agrupar código para reutilizar.

```
saludo() {  
  
    echo "Hola, $1"  
  
}  
  
saludo "Mundo"
```

Comandos comunes (grep, awk, sed, cut, sort, uniq): Procesamiento de texto y

```
datos. echo -e "manzana\nbanana\nmanzana" | sort | uniq
```

Pipes y redirecciones: Conectar la salida de un comando con la entrada de otro.

```
cat archivo.txt | grep "algo" > resultado.txt
```

Expresiones regulares: Patrones para buscar y manipular texto.

```
echo "error 404" | grep -Eo "[0-9]+"
```

Manejo de argumentos: Scripts que aceptan entrada del usuario.

```
#!/bin/bash  
  
echo "Ejecutando script con el argumento: $1"
```

Automatización y monitoreo: Scripts para automatizar tareas y monitorear

```
sistemas. #!/bin/bash  
  
if ps aux | grep -q "[a]pache2"; then  
  
    echo "Apache está corriendo."  
  
else  
  
    echo "Apache no está corriendo."  
  
fi
```

Procesamiento Paralelo con GNU Parallel: Ejecutar tareas en paralelo para optimizar el tiempo de procesamiento.

```
cat lista_urls.txt | parallel wget
```

Validación de entradas: Prevenir la ejecución de comandos maliciosos.

```
read -p "Introduce tu nombre: " nombre
```

echo "Hola, \$nombre" # Asegúrate de validar o escapar \$nombre si se usa en comandos más complejos.

Optimización de scripts: Utilizar herramientas y técnicas para reducir el tiempo de ejecución.

```
find . -name "*.txt" | xargs grep "patrón"
```

Ejercicios

Indica las actividades que realizan cada uno de los scripts (recuerda son archivos Bash y por tanto terminan en .sh).

```
#!/bin/bash
```

```
# Configuración
```

```
UMBRAL_CPU=70.0 # Uso máximo de CPU permitido (%)
```

```
UMBRAL_MEM=500 # Uso máximo de memoria permitido (MB)
```

```
LOG_FILE="/var/log/monitoreo_procesos.log"
```

```
EMAIL_ADMIN="admin@ejemplo.com"
```

```
PROCESOS_PARALELOS=("proceso1" "proceso2" "proceso3") # Nombres de los procesos a monitorear
```

```
# Función para convertir memoria de KB a MB
```

```
convertir_kb_a_mb() {
```

```
    echo "$(( $1 / 1024 ))"
```

```
}
```

```
# Función para obtener y verificar el uso de recursos de los procesos
```

```
verificar_procesos() {
```

```
    for PROC in "${PROCESOS_PARALELOS[@]}; do
```

```
        ps -C $PROC -o pid=,%cpu=,%mem=,vsz=,comm= --sort=-%cpu | while read PID CPU
```

```
MEM_VSZ_COMM; do
```

```
MEM_MB=$(convertir_kb_a_mb $VSZ)
```

```
if (( $(echo "$CPU > $UMBRAL_CPU" | bc -l) )) || [ "$MEM_MB" -gt  
"$UMBRAL_MEM" ]; then
```

```
echo "$(date +"%Y-%m-%d %H:%M:%S") - Proceso $COMM (PID $PID) excede los  
umbrales con CPU: $CPU%, MEM: ${MEM_MB}MB" >> $LOG_FILE
```

```
kill -9 $PID && echo "$(date +"%Y-%m-%d %H:%M:%S") - Proceso $PID  
terminado." >> $LOG_FILE
```

```
echo "Proceso $PID ($COMM) terminado por alto uso de recursos" | mail -s "Alerta  
de Proceso Terminado" $EMAIL_ADMIN
```

```
fi
```

```
done
```

```
done
```

```
}
```

```
# Loop principal para el monitoreo continuo
```

```
while true; do
```

```
verificar_procesos
```

```
sleep 60 # Espera 60 segundos antes de la próxima verificación
```

```
done
```

```

[1/2] ejercicio *
#!/bin/bash

# Configuración
UMBRAL_CPU=70.0
UMBRAL_MEM=500
LOG_FILE="/var/log/monitoreo_procesos.log"
EMAIL_ADMIN="admin@ejemplo.com"
PROCESOS_PARALELOS=("proceso1" "proceso2" "proceso3")

# Función para convertir memoria de KB a MB
convertir_kb_a_mb() {
    echo "$(( $1 / 1024 ))"
}

# Función para obtener y verificar el uso de recursos de los procesos
verificar_procesos() {
    for PROC in "${PROCESOS_PARALELOS[@]}; do
        # Ejecuta ps para obtener información sobre los procesos
        ps -C "$PROC" -o pid=%pid,%cpu=%cpu,%mem=%mem,vsz=,comm= --sort=-%cpu | while read PID CPU MEM VSZ COMM; do
            MEM_MB=$(convertir_kb_a_mb "$VSZ")
            # Verifica si el proceso supera los umbrales
            if (( $(echo "$CPU > $UMBRAL_CPU" | bc -l) )) || [ "$MEM_MB" -gt "$UMBRAL_MEM" ]; then
                # Registra en el log
                echo "$(date +%Y-%m-%d %H:%M:%S) - Proceso $COMM (PID $PID) excede los umbrales con CPU: $CPU%,>
                # Termina el proceso (con una señal menos drástica que SIGKILL)
                kill -TERM "$PID" && echo "$(date +%Y-%m-%d %H:%M:%S) - Proceso $PID terminado." >> "$LOG_FILE"
                # Envía alerta por correo electrónico
                echo "Proceso $PID ($COMM) terminado por alto uso de recursos" | mail -s "Alerta de Proceso Termini>
            fi
        done
    done
}

# Loop principal para el monitoreo continuo
while true; do

```

- Este script monitorea continuamente el uso de CPU y memoria de los procesos especificados en PROCESOS_PARALELOS. Cada minuto, verifica si alguno de estos procesos supera los umbrales de uso de CPU (UMBRAL_CPU) o de memoria (UMBRAL_MEM). Si un proceso excede alguno de estos umbrales, se registra en un archivo de registro (LOG_FILE), se detiene con kill -9 y se envía una alerta por correo electrónico al administrador (EMAIL_ADMIN)

.....

```

. #!/bin/bash

```

```
DIRECTORIOS=("dir1" "dir2" "dir3")
```

```
DESTINO_BACKUP="/mnt/backup"
```

```
backup_dir() {
```

```
    dir=$1
```

```
    fecha=$(date +%Y%m%d)
```

```
    tar -czf "${DESTINO_BACKUP}/${dir##*/}_${fecha}.tar.gz" "$dir"
```

```
    echo "Backup completado para $dir"
```

```
}
```

```
export -f backup_dir
```

```
export DESTINO_BACKUP
```

```
parallel backup_dir ::: "${DIRECTORIOS[@]}"
```

- **Este script realiza copias de seguridad de varios directorios en paralelo utilizando el comando tar. Cada directorio se comprime en un archivo .tar.gz con la fecha actual en el nombre y se guarda en un directorio de destino de backup. Utiliza el programa parallel para ejecutar las copias de seguridad en paralelo, lo que puede mejorar la eficiencia en sistemas con varios núcleos de CPU**

.....

```
. #!/bin/bash
```

```
NODOS=("nodo1" "nodo2" "nodo3")
```

```
TAREAS=("tarea1.sh" "tarea2.sh" "tarea3.sh")
```

```
distribuir_tareas() {
```

```
    for i in "${!TAREAS[@]"; do
```

```

nodo=${NODOS[$((i % ${#NODOS[@]}))]}

tarea=${TAREAS[$i]}

echo "Asignando $tarea a $nodo"

scp "$tarea" "${nodo}:/tmp"

ssh "$nodo" "bash /tmp/$tarea" &

done

wait

}

```

distribuir_tareas

- **Este script de distribuye tareas a nodos en una red. Utiliza un bucle para asignar cada tarea (TAREAS) a un nodo (NODOS). Utiliza scp para copiar la tarea al nodo y luego ejecuta la tarea en el nodo usando ssh. Las tareas se ejecutan en paralelo en los nodos. Finalmente, espera a que todas las tareas finalicen antes de continua**

```

#!/bin/bash

```

```

LOCK_FILE="/var/lock/mi_recurso.lock"

```

```

RECURSO="/path/to/recurso_compartido"

```

```

adquirir_lock() {

```

```

while ! (set -o noclobber; > "$LOCK_FILE") 2> /dev/null; do

```

```

echo "Esperando por el recurso..."

```

```

sleep 1

```

```

done

```

```
}
```

```
liberar_lock() {
```

```
rm -f "$LOCK_FILE"
```

```
}
```

```
adquirir_lock
```

```
# Trabajar con el recurso
```

```
echo "Accediendo al recurso"
```

```
sleep 5 # Simular trabajo
```

```
liberar_lock
```

- **Este script gestiona un bloqueo de recurso compartido utilizando un archivo de bloqueo (LOCK_FILE). La función adquirir_lock intenta adquirir el bloqueo del recurso, esperando si el bloqueo está en uso. Una vez adquirido el bloqueo, el script realiza alguna tarea con el recurso compartido (simulada con sleep 5 en este caso) y luego libera el bloqueo con la función liberar_lock, eliminando el archivo de bloqueo**

```
.....  
. #!/bin/bash
```

```
NODOS=("nodo1" "nodo2" "nodo3")
```

```
ARCHIVO_METRICAS="/tmp/metricas_$(date +%Y%m%d).csv"
```

```
recolectar_metricas() {
```

```
echo "Nodo,CPU(%),Memoria(%),Disco(%)" > "$ARCHIVO_METRICAS"
```

```
for nodo in "${NODOS[@]"; do
```

```
ssh "$nodo" "
```



```

cpu=$(top -bn1 | grep 'Cpu(s)' | sed 's/.*, *\[0-9.\]*\)%* id.*\1/' | awk '{print 100 - \$1}');

memoria=$(free | awk '/Mem:/ {print \$3/\$2 * 100.0}');

disco=$(df / | awk 'END{print $(NF-1)}');

echo "\"$HOSTNAME,\$cpu,\$memoria,\$disco\"";

" >> "$ARCHIVO_METRICAS"

done

}

recolectar_metricas

```

- **Recopila métricas de CPU, memoria y disco de varios nodos en una red. Utiliza un bucle para iterar sobre los nodos (NODOS) y, para cada nodo, se conecta mediante SSH para ejecutar comandos que obtienen las métricas de CPU, memoria y disco. Luego, las métricas se agregan a un archivo CSV (ARCHIVO_METRICAS) para su posterior análisis**

awk

Awk es una herramienta de scripting extremadamente poderosa y versátil para procesar y analizar datos en Unix/Linux. Es especialmente útil para manipular datos textuales y produce resultados formateados. **awk** funciona leyendo archivos o flujos de entrada línea por línea, dividiendo cada línea en campos, procesándola con acciones definidas por el usuario y luego imprimiendo la salida.

En el contexto de la computación paralela y concurrente, **awk** puede ser utilizado para analizar y procesar datos generados por procesos, monitorizar el rendimiento del sistema, y preparar datos para ser procesados en paralelo. Cuando se combina con pipes de Linux y expresiones regulares, **awk** se convierte en una herramienta aún más potente, permitiendo a los usuarios filtrar, procesar y redirigir la salida de comandos en secuencias complejas de operaciones.

awk puede ser usado para extraer información específica de la lista de procesos generada por el comando **ps**.

```
ps aux | awk '{print $1, $2, $3, $4, $11}' | head -n 10
```

Pregunta: ¿Qué hace y cual es el resultado del código anterior?

awk puede ser utilizado para preparar y filtrar datos que necesiten ser procesados en paralelo. Por ejemplo, puedes dividir un archivo grande en múltiples archivos más pequeños basados en algún criterio, que luego pueden ser procesados en paralelo:

```
awk '{print > ("output" int((NR-1)/1000) ".txt")}' input.txt
```

Pregunta: Comprueba con este archivo de texto el anterior script:

<https://babel.upm.es/~angel/teaching/pps/quijote.txt>

La combinación de **awk** con pipes y expresiones regulares expande significativamente sus capacidades de procesamiento de texto. Por ejemplo, para monitorizar archivos de log en busca de errores y filtrar mensajes relevantes:

```
tail -f /var/log/app.log | grep "ERROR" | awk '{print $1, $2, $NF}'
```

Pregunta: ¿puedes comprobar cual es el resultado si aplicas el script anterior en tus archivos logs?

Pregunta: ¿cuál es el resultado de utilizar este script (usa apache)?

```
ps -eo user,pid,pcpu,pmem,cmd | grep apache2 | awk '$3 > 50.0 || $4 > 50.0 {print "Alto recurso: ", $0}'
```

Ejercicios:

¿Cuál es la salida de los siguientes scripts (recuerda que son archivos de texto en bash)

1. `ps -eo pid,pcpu,pmem,cmd | awk '$2 > 10.0 || $3 > 10.0'`

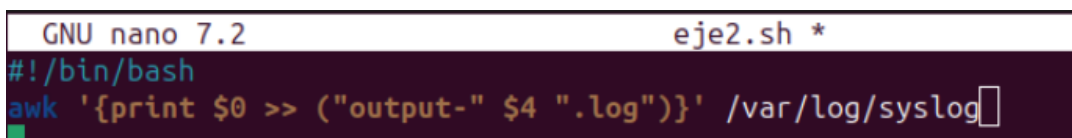


```
GNU nano 7.2                                     eje1.sh
#!/bin/bash

ps -eo pid,pcpu,pmem,cmd | awk '$2 > 10.0 || $3 > 10.0'

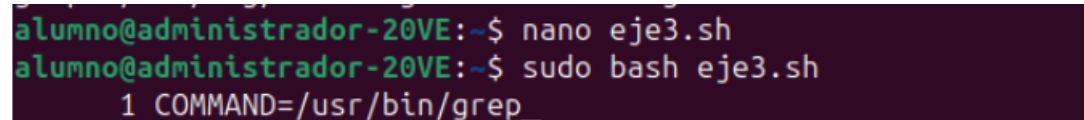
alumno@administrador-20VE:~$ bash eje1.sh
4538  2.4  1.7 /usr/bin/gnome-shell
5920  8.2  6.3 /snap/firefox/4090/usr/lib/firefox/firefox
6449  0.9  2.3 /snap/firefox/4090/usr/lib/firefox/firefox -contentproc
6974  2.8  5.8 /snap/firefox/4090/usr/lib/firefox/firefox -contentproc
10836 3.2  2.1 /snap/firefox/4090/usr/lib/firefox/firefox -contentproc
22852 6.5  1.4 /snap/firefox/4090/usr/lib/firefox/firefox -contentproc
```

2. `awk '{print $0 >> ("output-" $4 ".log")}' /var/log/syslog`



```
GNU nano 7.2                                     eje2.sh *
#!/bin/bash
awk '{print $0 >> ("output-" $4 ".log")}' /var/log/syslog
```

3. `grep "Failed password" /var/log/auth.log | awk '{print $(NF-3)}' | sort | uniq -c | sort -nr`



```
alumno@administrador-20VE:~$ nano eje3.sh
alumno@administrador-20VE:~$ sudo bash eje3.sh
1 COMMAND=/usr/bin/grep
```

4. `inotifywait -m /path/to/dir -e create | awk '{print "Nuevo archivo creado:", $3}'` 5. `find . -type f -name "*.py" -exec ls -l {} + | awk '{sum += $5} END {print "Espacio total usado por archivos .py: ", sum}'`

6. `awk '{sum+=$NF} END {print "Tiempo promedio de respuesta:", sum/NR}' access.log`

7. `ps -eo state | awk '/D/ {d++} /R/ {r++} END {print "Espera (D):", d, "- Ejecución (R):", r}'`

r}'

```
alumno@administrador-20VE:~$ ps -eo state | awk '/D/ {d++} /R/ {r++} END {print "Espera (D):", d, "- Ejecución (R):", r}'  
Espera (D): 1 - Ejecución (R): 1
```

8. ps -eo state | awk '/D/ {d++} /R/ {r++} END {print "Espera (D):", d, "- Ejecución (R):", r}'

```
alumno@administrador-20VE:~$ ps -eo state | awk '/D/ {d++} /R/ {r++} END {print "Espera (D):", d, "- Ejecución (R):", r}'  
Espera (D): 1 - Ejecución (R): 1
```

9. awk '/SwapTotal/ {total=\$2} /SwapFree/ {free=\$2} END {if ((total-free)/total*100 > 20.0) print "Alerta: Uso excesivo de swap"}' /proc/meminfo

```
alumno@administrador-20VE:~$ awk '/SwapTotal/ {total=$2} /SwapFree/ {free=$2} END {if ((total-free)/total*100 > 20.0) print "Alerta: Uso excesivo de swap"}' /proc/meminfo
```

10. ls -l | awk '!/^total/ && !/^d/ {sum += \$5} END {print "Uso total de disco (sin subdirectorios):", sum}'

```
alumno@administrador-20VE:~$ ls -l | awk '!/^total/ && !/^d/ {sum += $5} END {print "Uso total de disco (sin subdirectorios):", sum}'  
Uso total de disco (sin subdirectorios): 34837261
```

