

## 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

```

alumno@administrador-20VE:~$ ps
  PID TTY          TIME CMD
 22240 pts/0    00:00:00 bash
 22296 pts/0    00:00:00 ps
alumno@administrador-20VE:~$ ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1   0.0  0.0 168968 12884 ?        Ss   08:39   0:07 /sbin/init sp
root         2   0.0  0.0      0     0 ?        S    08:39   0:00 [kthreadd]
root         3   0.0  0.0      0     0 ?        I<   08:39   0:00 [rcu_gp]
root         4   0.0  0.0      0     0 ?        I<   08:39   0:00 [rcu_par_gp]
root         5   0.0  0.0      0     0 ?        I<   08:39   0:00 [slub_flushwq
root         6   0.0  0.0      0     0 ?        I<   08:39   0:00 [netns]
root         8   0.0  0.0      0     0 ?        I<   08:39   0:00 [kworker/0:0H
root        10   0.0  0.0      0     0 ?        I<   08:39   0:00 [mm_percpu_wq
root        11   0.0  0.0      0     0 ?        I    08:39   0:00 [rcu_tasks_kt
root        12   0.0  0.0      0     0 ?        I    08:39   0:00 [rcu_tasks_ru
root        13   0.0  0.0      0     0 ?        I    08:39   0:00 [rcu_tasks_tr
root        14   0.0  0.0      0     0 ?        S    08:39   0:00 [ksoftirqd/0]
root        15   0.0  0.0      0     0 ?        I    08:39   0:05 [rcu_preempt]
root        16   0.0  0.0      0     0 ?        S    08:39   0:00 [migration/0]
root        17   0.0  0.0      0     0 ?        S    08:39   0:00 [idle_inject/
root        19   0.0  0.0      0     0 ?        S    08:39   0:00 [cpuhp/0]
root        20   0.0  0.0      0     0 ?        S    08:39   0:00 [cpuhp/1]
root        21   0.0  0.0      0     0 ?        S    08:39   0:00 [idle_inject/
root        22   0.0  0.0      0     0 ?        S    08:39   0:00 [migration/1]
root        23   0.0  0.0      0     0 ?        S    08:39   0:00 [ksoftirqd/1]
root        25   0.0  0.0      0     0 ?        I<   08:39   0:00 [kworker/1:0H
root        26   0.0  0.0      0     0 ?        S    08:39   0:00 [cpuhp/2]
root        27   0.0  0.0      0     0 ?        S    08:39   0:00 [idle_inject/
root        28   0.0  0.0      0     0 ?        S    08:39   0:00 [migration/2]

```

## 2. Buscar procesos específicos por nombre:

```

alumno@administrador-20VE:~$ ps aux | grep firefox
alumno    5920  7.2  6.0 30161376 976744 ?        Sl   09:05  13:10 /snap/firefox
alumno    6107  0.0  0.2 209900 48384 ?        Sl   09:05   0:00 /snap/firefox
alumno    6143  0.0  0.7 2476772 128808 ?        Sl   09:05   0:04 /snap/firefox
alumno    6265  0.0  0.6 2443944 103184 ?        Sl   09:05   0:00 /snap/firefox
alumno    6439  0.0  0.3 344512 58988 ?        Sl   09:05   0:00 /snap/firefox
alumno    6449  1.2  2.3 2808760 377612 ?        Sl   09:05   2:18 /snap/firefox
alumno    6587  0.0  0.7 595108 120896 ?        Sl   09:05   0:00 /snap/firefox
alumno    6974  1.8  6.0 3718908 976808 ?        Sl   09:08   3:15 /snap/firefox
alumno    8910  0.6  0.9 2605260 154504 ?        Sl   09:29   1:00 /snap/firefox
alumno    9628  0.1  0.9 2496716 157012 ?        Sl   09:37   0:09 /snap/firefox
alumno   10836  5.2  2.1 2831468 340340 ?        Sl   09:50   7:07 /snap/firefox
alumno   12769  0.0  0.6 2433912 100736 ?        Sl   10:11   0:01 /snap/firefox
alumno   12888  0.0  0.7 2442848 117872 ?        Sl   10:12   0:04 /snap/firefox
alumno   13749  0.0  0.8 2472396 129348 ?        Sl   10:21   0:02 /snap/firefox
alumno   16075  0.0  0.7 2465108 115012 ?        Sl   10:49   0:01 /snap/firefox
alumno   18771  0.0  0.6 2456352 109908 ?        Sl   11:20   0:01 /snap/firefox
alumno   18830  0.2  1.0 2496980 163388 ?        Sl   11:20   0:05 /snap/firefox
alumno   20539  0.1  1.2 2575984 202324 ?        Sl   11:41   0:02 /snap/firefox
alumno   22041  1.1  1.2 2580208 205392 ?        Sl   11:56   0:06 /snap/firefox
alumno   22110  0.0  0.6 2424076 97632 ?        Sl   11:56   0:00 /snap/firefox
alumno   22801  0.5  0.9 2507292 160572 ?        Sl   12:03   0:00 /snap/firefox
alumno   22852  5.6  1.2 6855736 209016 ?        Sl   12:03   0:09 /snap/firefox
alumno   22887  0.0  0.5 2416944 84376 ?        Sl   12:03   0:00 /snap/firefox
alumno   23025  0.0  0.4 2390244 72064 ?        Sl   12:04   0:00 /snap/firefox
alumno   23079  0.0  0.4 2390244 73472 ?        Sl   12:04   0:00 /snap/firefox
alumno   23082  0.0  0.4 2390240 71424 ?        Sl   12:04   0:00 /snap/firefox
alumno   23269  0.0  0.0   6576  2432 pts/0    S+   12:06   0:00 grep --color=
alumno@administrador-20VE:~$

```

3. Mostrar procesos en un árbol jerárquico (útil para ver relaciones padre-hijo en procesos concurrentes):

```
alumno@administrador-20VE: ~  
alumno@administrador-20VE:~$ ps -ejH
```

| PID | PGID | SID | TTY | TIME     | CMD                         |
|-----|------|-----|-----|----------|-----------------------------|
| 2   | 0    | 0   | ?   | 00:00:00 | kthreadd                    |
| 3   | 0    | 0   | ?   | 00:00:00 | rcu_gp                      |
| 4   | 0    | 0   | ?   | 00:00:00 | rcu_par_gp                  |
| 5   | 0    | 0   | ?   | 00:00:00 | slub_flushwq                |
| 6   | 0    | 0   | ?   | 00:00:00 | netns                       |
| 8   | 0    | 0   | ?   | 00:00:00 | kworker/0:0H-events_highpri |
| 10  | 0    | 0   | ?   | 00:00:00 | mm_percpu_wq                |
| 11  | 0    | 0   | ?   | 00:00:00 | rcu_tasks_kthread           |
| 12  | 0    | 0   | ?   | 00:00:00 | rcu_tasks_rude_kthread      |
| 13  | 0    | 0   | ?   | 00:00:00 | rcu_tasks_trace_kthread     |
| 14  | 0    | 0   | ?   | 00:00:00 | ksoftirqd/0                 |
| 15  | 0    | 0   | ?   | 00:00:05 | rcu_preempt                 |
| 16  | 0    | 0   | ?   | 00:00:00 | migration/0                 |
| 17  | 0    | 0   | ?   | 00:00:00 | idle_inject/0               |
| 19  | 0    | 0   | ?   | 00:00:00 | cpuhp/0                     |
| 20  | 0    | 0   | ?   | 00:00:00 | cpuhp/1                     |
| 21  | 0    | 0   | ?   | 00:00:00 | idle_inject/1               |
| 22  | 0    | 0   | ?   | 00:00:00 | migration/1                 |
| 23  | 0    | 0   | ?   | 00:00:00 | ksoftirqd/1                 |
| 25  | 0    | 0   | ?   | 00:00:00 | kworker/1:0H-events_highpri |
| 26  | 0    | 0   | ?   | 00:00:00 | cpuhp/2                     |
| 27  | 0    | 0   | ?   | 00:00:00 | idle_inject/2               |
| 28  | 0    | 0   | ?   | 00:00:00 | migration/2                 |
| 29  | 0    | 0   | ?   | 00:00:00 | ksoftirqd/2                 |
| 31  | 0    | 0   | ?   | 00:00:00 | kworker/2:0H-events_highpri |
| 32  | 0    | 0   | ?   | 00:00:00 | cpuhp/3                     |
| 33  | 0    | 0   | ?   | 00:00:00 | idle_inject/3               |
| 34  | 0    | 0   | ?   | 00:00:00 | migration/3                 |

4. Mostrar procesos de un usuario específico:

```

alumno@administrador-20VE:~$ ps -u alumno
  PID TTY          TIME CMD
  4180 ?            00:00:00 systemd
  4181 ?            00:00:00 (sd-pam)
  4188 ?            00:00:00 pipewire
  4191 ?            00:00:00 ubuntu-report
  4195 ?            00:00:00 wireplumber
  4203 ?            00:00:00 pipewire-pulse
  4206 ?            00:00:00 gnome-keyring-d
  4207 ?            00:00:00 dbus-daemon
  4233 ?            00:00:00 xdg-document-po
  4237 ?            00:00:00 xdg-permission-
  4280 tty2        00:00:00 gdm-x-session
  4283 tty2        00:02:46 Xorg
  4385 tty2        00:00:00 gnome-session-b
  4471 ?            00:00:00 at-spi-bus-laun
  4478 ?            00:00:00 dbus-daemon
  4492 ?            00:00:00 gcr-ssh-agent
  4493 ?            00:00:00 gnome-session-c
  4495 ?            00:00:00 ssh-agent
  4505 ?            00:00:00 gvfsd
  4511 ?            00:00:00 gvfsd-fuse
  4513 ?            00:00:00 gnome-session-b
  4538 ?            00:03:47 gnome-shell
  4570 ?            00:00:01 mutter-x11-fram
  4586 ?            00:00:00 snapd-desktop-i
  4671 ?            00:00:00 snapd-desktop-i
  4675 ?            00:00:00 xdg-desktop-por
  4680 ?            00:00:01 xdg-desktop-por
  4711 ?            00:00:00 gnome-shell-cal
  4721 ?            00:00:00 evolution-sourc
  4757 ?            00:00:00 gcr-daemon

```

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

```

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

nombre="firefox"

if ps aux | grep -v grep | grep "$nombre" >/dev/null; then
    echo "El proceso $nombre está en ejecución."
else
    echo "El proceso $nombre no está en ejecución. Reiniciando..."
fi

```

```

alumno@administrador-20VE:~$ bash verifica.sh
El proceso firefox está en ejecución.

```

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

```

```

(standard_in) 1: syntax error
alumno@administrador-20VE:~$ nano monitoreo.sh
alumno@administrador-20VE:~$ bash monitoreo.sh
(standard_in) 1: syntax error
Proceso 7341 (ps -eo pid,ppid,%cpu,cmd --sort=-%cpu) está utilizando 100% de CPU.

```

### 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

```

```

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

```

### 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

```

```

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

```

### 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

```



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

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

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

```
ps -eo uid,pid,cmd | awk '$1 > 1000 {print}'
```

```
alumno@administrador-20VE:~$ bash proces.sh
UID      PID  CMD
1001     2100 /lib/systemd/systemd --user
1001     2101 (sd-pam)
1001     2108 /usr/bin/pipewire
1001     2111 /usr/bin/ubuntu-report service
1001     2112 /usr/bin/wireplumber
1001     2124 /usr/bin/pipewire-pulse
1001     2126 /usr/bin/gnome-keyring-daemon --foreground --components=pkcs11,secrets --control-directory=/run/user/1001/keyring
1001     2127 /usr/bin/dbus-daemon --session --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
1001     2153 /usr/libexec/xdg-document-portal
1001     2157 /usr/libexec/xdg-permission-store
1001     2233 /usr/libexec/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --session=ubuntu
1001     2235 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1001/gdm/Xauthority -nolisten tcp -background none -noreset -keeptty
1001     2315 /usr/libexec/gnome-session-binary --session=ubuntu
1001     2335 /snap/snapd-desktop-integration/83/usr/bin/snapd-desktop-integration
1001     2467 /snap/snapd-desktop-integration/83/usr/bin/snapd-desktop-integration
1001     2475 /usr/libexec/at-spi-bus-launcher
1001     2482 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --print-address 11 --address=
1001     2503 /usr/libexec/gcr-ssh-agent /run/user/1001/gcr
1001     2504 /usr/libexec/gnome-session-ctl --monitor
1001     2506 ssh-agent -D -a /run/user/1001/openssh_agent
1001     2516 /usr/libexec/gvfsd
1001     2524 /usr/libexec/gvfsd-fuse /run/user/1001/gvfs -f
1001     2527 /usr/libexec/gnome-session-binary --systemd-service --session=ubuntu
1001     2533 /usr/libexec/xdg-desktop-portal
1001     2543 /usr/libexec/xdg-desktop-portal-gnome
1001     2560 /usr/bin/gnome-shell
1001     2611 /usr/libexec/xdg-desktop-portal-gtk
1001     2626 /usr/libexec/at-spi2-registryd --use-gnome-session
1001     2628 /usr/bin/gjs /usr/share/gnome-shell/org.gnome.ScreenSaver
1001     2658 /usr/libexec/mutter-x11-frames
1001     2708 /usr/libexec/gnome-shell-calendar-server
1001     2715 /usr/libexec/evolution-source-registry
1001     2743 /usr/libexec/dconf-service
```

## 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."
```

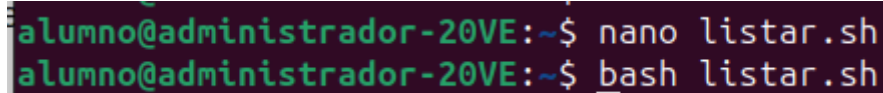
```
fi
```

```
done
```

```
GNU nano 7.2 alerta.sh
#!/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
```

## 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
```



A terminal window showing the execution of the script. The prompt is 'alumno@administrador-20VE:~\$'. The user enters 'nano listar.sh' and then 'bash listar.sh'.

## 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
```



A terminal window showing the execution of the script. The prompt is 'GNU nano 7.2'. The user enters 'monitoriar.sh'. The script content is visible in the terminal, including the shebang, variable assignments, and the monitoring loop.

## 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."
```



```

alumno@administrador-20VE:~$ bash inform.sh
Informe generado en proceso_informe.txt.
alumno@administrador-20VE:~$ cat proceso_informe.txt
  PID      ELAPSED  CMD
    1      01:24:51 /sbin/init splash
    2      01:24:51 [kthreadd]
    3      01:24:51 [rcu_gp]
    4      01:24:51 [rcu_par_gp]
    5      01:24:51 [slub_flushwq]
    6      01:24:51 [netns]
    8      01:24:51 [kworker/0:0H-events_highpri]
   10      01:24:51 [mm_percpu_wq]
   11      01:24:51 [rcu_tasks_kthread]
   12      01:24:51 [rcu_tasks_rude_kthread]
   13      01:24:51 [rcu_tasks_trace_kthread]
   14      01:24:51 [ksoftirqd/0]
   15      01:24:51 [rcu_preempt]
   16      01:24:51 [migration/0]
   17      01:24:51 [idle_inject/0]
   19      01:24:51 [cpuhp/0]
   20      01:24:51 [cpuhp/1]
   21      01:24:51 [idle_inject/1]
   22      01:24:51 [migration/1]
alumno@administrador-20VE:~$ 

```

## 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 && mail
-s "Alerta de Memoria" alumno@administrador-20VE < /dev/null
alumno@administrador-20VE:~$
```

### 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
```

Monitorea los errores de conexión luego envía esas líneas por correo electrónico al administrador con la dirección admin@example.com, notificándole sobre los errores de conexión detectados.

### 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, $NF}' | sort
alumno@administrador-20VE:~$
```

### 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
alumno@administrador-20VE:~$
```

```
Cada 5,0s: ps aux | grep -vE "(root|daemon|nobody)" | gre... administrador-20VE: Mon Apr 15 14:34:32 2024
```

| 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-oomd               |
| systemd+ | 561  | 0.0  | 0.0  | 20488    | 12416  | ?    | Ss   | 12:43 | 0:02 | /lib/systemd/systemd-resolved           |
| systemd+ | 562  | 0.0  | 0.0  | 89692    | 7296   | ?    | Ssl  | 12:43 | 0:00 | /lib/systemd/systemd-timesyncd          |
| 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-persistenced --user nvi |
| colord   | 1368 | 0.0  | 0.0  | 316520   | 12924  | ?    | Ssl  | 12:43 | 0:00 | /usr/libexec/colord                     |
| cups-br+ | 1693 | 0.0  | 0.0  | 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 --test             |
| 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 --user             |
| 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 service          |
| 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-document-portal        |
| alumno   | 2157 | 0.0  | 0.0  | 307088   | 6144   | ?    | Ssl  | 12:44 | 0:00 | /usr/libexec/xdg-permission-store       |
| alumno   | 2233 | 0.0  | 0.0  | 233252   | 6144   | tty2 | Ssl+ | 12:44 | 0:00 | /usr/libexec/gdm-x-session --run-script |
| alumno   | 2235 | 2.1  | 0.9  | 26897084 | 151840 | tty2 | Sl+  | 12:44 | 2:23 | /usr/lib/xorg/Xorg vt2 -displayfd 3 -au |

## 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 ;
alumno@administrador-20VE:~$
```

## 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
```

```
GNU nano 7.2 noautor.sh
#!/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}'"
```

```
GNU nano 7.2                                ejercicio1.sh *
#!/bin/bash

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

```
alumno@administrador-20VE: ~
Cada 2,0s: ps aux | grep '[a]pache2' | awk '{print $1, $2, $3, $4, $11}'  administrador-20VE: Mon Apr 15 14:42:44 2024
```

```
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
alumno@administrador-20VE:~$
```

```
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.
alumno@administrador-20VE:~$
```

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

Detectar procesos que estén utilizando más del 80% de la CPU en el sistema y enviar una alerta por correo electrónico al administrador del sistema.

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

Buscar la cadena "ERROR" en todos los archivos con extensión ".log" y guardar los resultados en archivos separados 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 "$(date)
Fallo de ping" >> fallos_ping.txt
64 bytes from 8.8.8.8: icmp_seq=1 ttl=54 time=36.2 ms
64 bytes from 93.184.216.34 (93.184.216.34): icmp_seq=1 ttl=52 time=8
0.5 ms
alumno@administrador-20VE:~$
```

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

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

```
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")

**Identifica sistemas de archivos en el directorio /home con más del 80% de**



**espacio utilizado, crea archivos tar.gz para ellos y luego verifica el uso de CPU de los procesos, emitiendo una alerta si algún proceso excede el 80% de 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")
```

```
# 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
```

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

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

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

**Filtra las líneas con la palabra "ERROR" en un archivo, cuenta las ocurrencias de cada tipo de error y luego imprime el recuento de errores.**

```
.....

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')

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

# Lista de archivos divididos

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

# 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))
```

**Analiza cada parte del archivo en paralelo utilizando múltiples procesos y finalmente imprime el total de errores encontrados en todo el archivo de registro.**

```
.....

. import subprocess

import time
```

```
previous_ports = set()
```

```
while True:
```

```
# 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)
```

```
# 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
```

**Monitorea los cambios en los puertos abiertos en el sistema cada minuto y muestra cualquier nuevo puerto abierto o puerto cerrado.**

.....

```
import subprocess
```

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

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

```

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

# 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")

```

**Muestra la información sobre el uso de memoria por proceso y calcula el uso total de memoria por usuario.**

```

.....

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)
```

**Recopila datos sobre el uso de CPU y memoria en intervalos regulares y los guarda en un archivo.**

```
.....  
#!/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
```

**Monitorea continuamente el uso de CPU por parte de los procesos y envía alertas.**

```
.....  
.....  
#!/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."
```

**Guarda en un archivo de texto el uso de la memoria por usuario, pero  
ordenada según la cantidad total de memoria utilizada.**

```
.....  
#!/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
```

**Muestra una lista de los usuarios con el mayor uso de CPU y memoria en el sistema.**

```
.....  
. #!/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 "%"}'
```

Muestra un informe de memoria para los procesos que se llaman "java", incluyendo  
información sobre el PID, el usuario, el porcentaje de uso de memoria.

```
.....  
. #!/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=$(geoiplookup $IP | cut -d, -f2)
```

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

```
done
```

### **Muestra las 5 direcciones IP principales que acceden al servidor web**

.....

```
#!/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"
```

**Muestra la tasa de transferencia de datos de recepción y transmisión para una interfaz de red durante 5 segundos.**

## **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>

## Computación Paralela y Distribuida

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
```

## Computación Paralela y Distribuida

**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=,%cpu=,%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
```

**Monitorea continuamente el uso de CPU y memoria de una lista específica de procesos.**

**Cuando uno de estos procesos excede los umbrales de uso de CPU envía una alerta al email del administrador.**

---

```
. #!/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[@]}"
```

**Realiza copias de seguridad (backups) de los directorios "dir1" "dir2" "dir3".**

---

```
. #!/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

- **Distribuye la tareas ("tarea1.sh" "tarea2.sh" "tarea3.sh") a través de los nodos (nodo1" "nodo2" "nodo3" )en una red.**

---

```

. #!/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
```

- **Gestiona un recurso compartido mediante un mecanismo de bloqueo para evitar que múltiples procesos accedan simultáneamente al recurso.**

```
.....  
  
. #!/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
```

**Recolecta métricas de rendimiento de CPU, memoria y disco de varios nodos remotos y las guarda en un archivo.**

## **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 -
alumno@administrador-20VE:~$
```

Muestra los procesos que están utilizando más del 10% de la CPU o más del 10% de la memoria en el sistema.

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
```

Divide el archivo /var/log/syslog en varios archivos de salida.

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
alumno@administrador-20VE:~$

```

Encuentra todas las entradas en el archivo de registro de autenticación que indican intentos fallidos de inicio de sesión.

4. `inotifywait -m /path/to/dir -e create | awk '{print "Nuevo archivo creado:", $3}'`

Muestra en tiempo real los nombres de los archivos que se crean en el directorio /path/to/dir

5. `find . -type f -name "*.py" -exec ls -l {} + | awk '{sum += $5} END {print "Espacio total usado por archivos .py: ", sum}'`

Muestra el tamaño total en bytes de todos los archivos .py encontrados en el directorio actual y sus subdirectorios.

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

7. `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): - Ejecución (R): 1
alumno@administrador-20VE:~$

```

Muestra el número de procesos en espera y en ejecución.

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
alumno@administrador-20VE:~$

```

Muestra el número de procesos en espera y en ejecución.

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

```

Monitorea el uso de swap en el sistema y emite una alerta si el porcentaje de uso de swap es superior al 20%.

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 += $5} END {pr  
int "Uso total de disco (sin subdirectorios):", sum}'  
Uso total de disco (sin subdirectorios): 34837261  
alumno@administrador-20VE:~$
```

Muestra el uso total de disco de todos los archivos en el directorio actual,  
excluyendo los subdirectorios.