

PARTE N°1

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:

Ejercicios

1. Listar todos los procesos con detalles completos
2. Buscar procesos específicos por nombre:
3. Mostrar procesos en un árbol jerárquico (útil para ver relaciones padre-hijo en procesos concurrentes):
4. Mostrar procesos de un usuario específico:
5. Escribe un script para verificar y reiniciar automáticamente un proceso si no está corriendo.

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

1. 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
```

Este script de bash primero obtiene los 10 procesos con mayor uso de CPU. Luego para cada uno de estos procesos, verifica si el uso de CPU es mayor al 80%. Como en este caso no existe proceso mayor a 80% por ello no imprime nada.

```
alumno@administrador-20VE: ~  
alumno@administrador-20VE:~$ touch bash1.sh  
alumno@administrador-20VE:~$ nano bash1.sh  
alumno@administrador-20VE:~$ chmod +x bash1.sh  
alumno@administrador-20VE:~$ ./bash1.sh  
alumno@administrador-20VE:~$
```

2. 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 script2.sh  
Proceso zombi detectado: PID=14846 CMD=[sd_espeak-ng-mb] <defunct>
```

Por lo tanto, este script detecta procesos zombis y reporta su PID y el comando que los inició. Un proceso zombi es un proceso que ha completado su ejecución pero aún tiene una entrada en la tabla de procesos, permitiendo que el proceso padre lea su estado de salida. En la mayoría de los casos, los procesos zombis no son problemáticos, pero si hay muchos pueden agotar la tabla de procesos.

El PID (ID de proceso) de este proceso zombi es 14846 y el comando que lo inició es [sd_espeak-ng-mb]. El término <defunct> es otro indicador de que este proceso es un proceso zombi.

3. 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 script3.sh  
alumno@administrador-20VE:~$ bash script3.sh  
Failed to restart apache2.service: Access denied  
See system logs and 'systemctl status apache2.service' for details.  
apache2 ha sido reiniciado.
```

Por lo tanto, si el servicio “apache2” no está en ejecución cuando se ejecuta este script, debería imprimir “apache2 ha sido reiniciado.”

4. 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
```

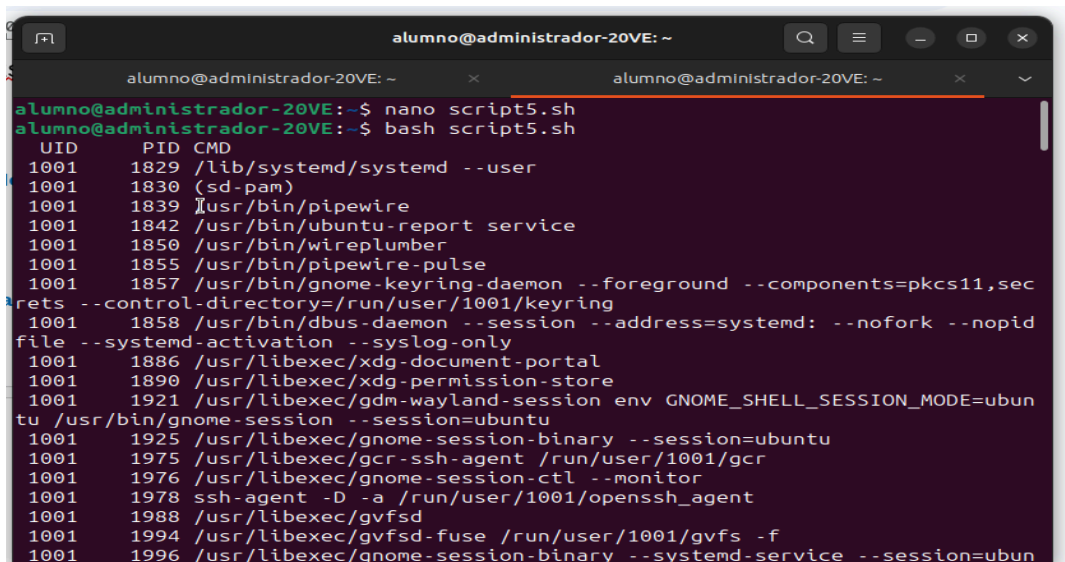
```
alumno@administrador-20VE:~$ touch bash4.sh  
alumno@administrador-20VE:~$ nano bash4.sh  
alumno@administrador-20VE:~$ chmod +x bash4.sh  
alumno@administrador-20VE:~$ ./bash4.sh  
alumno@administrador-20VE:~$
```

El script que has escrito está diseñado para imprimir un mensaje solo cuando el número de instancias del proceso “httpd” supera el valor de **MAX_INSTANCES**, que en tu caso es 10. Si no imprime nada, significa que el número de instancias de “httpd” es menor o igual a 10.

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

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

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



```
alumno@administrador-20VE: ~  
alumno@administrador-20VE: ~  
alumno@administrador-20VE:~$ nano script5.sh  
alumno@administrador-20VE:~$ bash script5.sh  
UID      PID  CMD  
1001     1829 /lib/systemd/systemd --user  
1001     1830 (sd-pam)  
1001     1839 /usr/bin/pipewire  
1001     1842 /usr/bin/ubuntu-report service  
1001     1850 /usr/bin/wireplumber  
1001     1855 /usr/bin/pipewire-pulse  
1001     1857 /usr/bin/gnome-keyring-daemon --foreground --components=pkcs11,sec  
rets --control-directory=/run/user/1001/keyring  
1001     1858 /usr/bin/dbus-daemon --session --address=systemd: --nofork --nopid  
file --systemd-activation --syslog-only  
1001     1886 /usr/libexec/xdg-document-portal  
1001     1890 /usr/libexec/xdg-permission-store  
1001     1921 /usr/libexec/gdm-wayland-session env GNOME_SHELL_SESSION_MODE=ubun  
tu /usr/bin/gnome-session --session=ubuntu  
1001     1925 /usr/libexec/gnome-session-binary --session=ubuntu  
1001     1975 /usr/libexec/gcr-ssh-agent /run/user/1001/gcr  
1001     1976 /usr/libexec/gnome-session-ctl --monitor  
1001     1978 ssh-agent -D -a /run/user/1001/openssh_agent  
1001     1988 /usr/libexec/gvfsd  
1001     1994 /usr/libexec/gvfsd-fuse /run/user/1001/gvfs -f  
1001     1996 /usr/libexec/gnome-session-binary --systemd-service --session=ubun
```

Por lo tanto, este script imprimirá una lista de todos los procesos que están siendo ejecutados por usuarios con un UID mayor a 1000, lo que generalmente corresponde a usuarios sin privilegios en un sistema Linux. Cada línea de la salida incluirá el UID del usuario que está ejecutando el proceso, el PID del proceso y el comando que inició el proceso.

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/://')

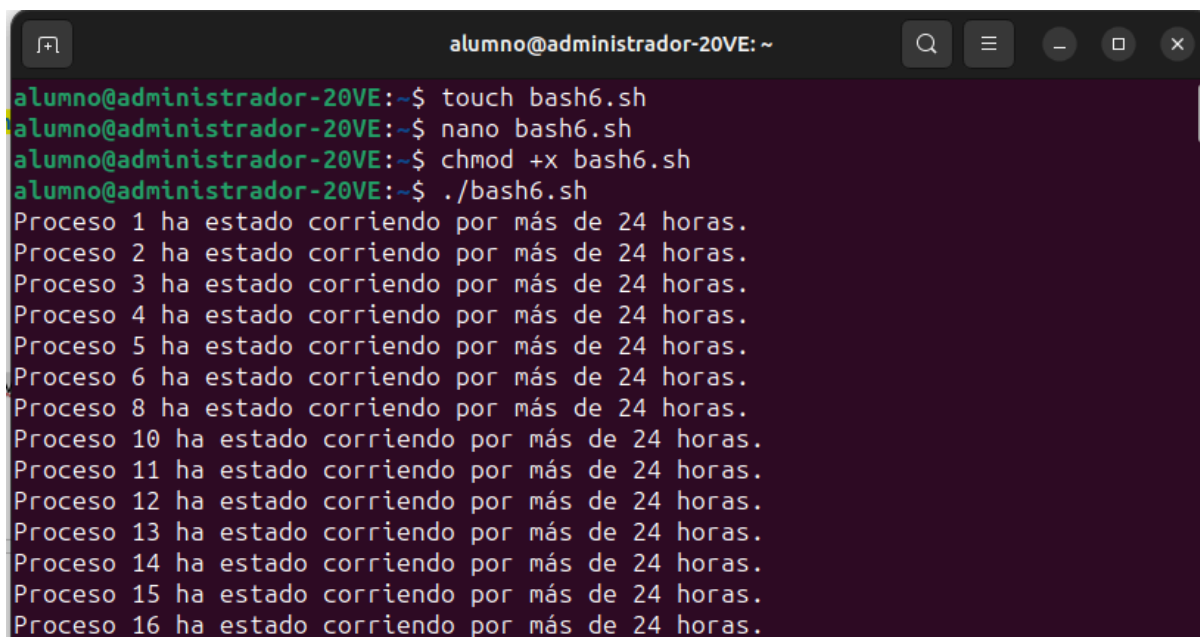
    total_hours=$((days * 24 + hours))

    if [ $total_hours -gt $MAX_HOURS ]; then

        echo "Proceso $pid ha estado corriendo por más de $MAX_HOURS horas."

    fi

done
```

A terminal window titled 'alumno@administrador-20VE: ~' with standard window controls. The user enters a series of commands: 'touch bash6.sh', 'nano bash6.sh', 'chmod +x bash6.sh', and './bash6.sh'. The output of the script is displayed, showing 16 lines of messages: 'Proceso 1 ha estado corriendo por más de 24 horas.' through 'Proceso 16 ha estado corriendo por más de 24 horas.'.

```
alumno@administrador-20VE:~$ touch bash6.sh
alumno@administrador-20VE:~$ nano bash6.sh
alumno@administrador-20VE:~$ chmod +x bash6.sh
alumno@administrador-20VE:~$ ./bash6.sh
Proceso 1 ha estado corriendo por más de 24 horas.
Proceso 2 ha estado corriendo por más de 24 horas.
Proceso 3 ha estado corriendo por más de 24 horas.
Proceso 4 ha estado corriendo por más de 24 horas.
Proceso 5 ha estado corriendo por más de 24 horas.
Proceso 6 ha estado corriendo por más de 24 horas.
Proceso 8 ha estado corriendo por más de 24 horas.
Proceso 10 ha estado corriendo por más de 24 horas.
Proceso 11 ha estado corriendo por más de 24 horas.
Proceso 12 ha estado corriendo por más de 24 horas.
Proceso 13 ha estado corriendo por más de 24 horas.
Proceso 14 ha estado corriendo por más de 24 horas.
Proceso 15 ha estado corriendo por más de 24 horas.
Proceso 16 ha estado corriendo por más de 24 horas.
```

Por lo tanto, este script imprimirá un mensaje para cada proceso que ha estado en ejecución durante más de 24 horas, indicando el PID del proceso y que ha estado en ejecución durante más de 24 horas. Si no hay procesos que hayan estado en ejecución durante más de 24 horas, el script no imprimirá nada.

7. 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

```
alumno@administrador-20VE:~$ nano script7.sh
alumno@administrador-20VE:~$ bash script7.sh
  PID CMD
 2749 /snap/firefox/3836/usr/lib/firefox/firefox
  PID CMD
 2749 /snap/firefox/3836/usr/lib/firefox/firefox
alumno@administrador-20VE:~$
```

Por lo tanto, este script imprimirá una lista de todos los procesos que están escuchando en el puerto 80, indicando el PID del proceso y el comando que lo inició. Si no hay procesos que estén escuchando en el puerto 80, el script no imprimirá nada.

8. 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

```
alumno@administrador-20VE:~$ touch bash8.sh
alumno@administrador-20VE:~$ nano bash8.sh
alumno@administrador-20VE:~$ chmod +x bash8.sh
alumno@administrador-20VE:~$ ./bash8.sh
alumno@administrador-20VE:~$
```

Si el script no imprime nada, es posible que no haya ningún proceso “mysqld” que esté utilizando más de 1GB de memoria en tu sistema. El script está diseñado para imprimir un mensaje solo cuando encuentra un proceso que cumple con esa condición.

9. 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:~$ nano script9.sh
alumno@administrador-20VE:~$ bash script9.sh
Informe generado en proceso_informe.txt.
alumno@administrador-20VE:~$

```

El comando grep

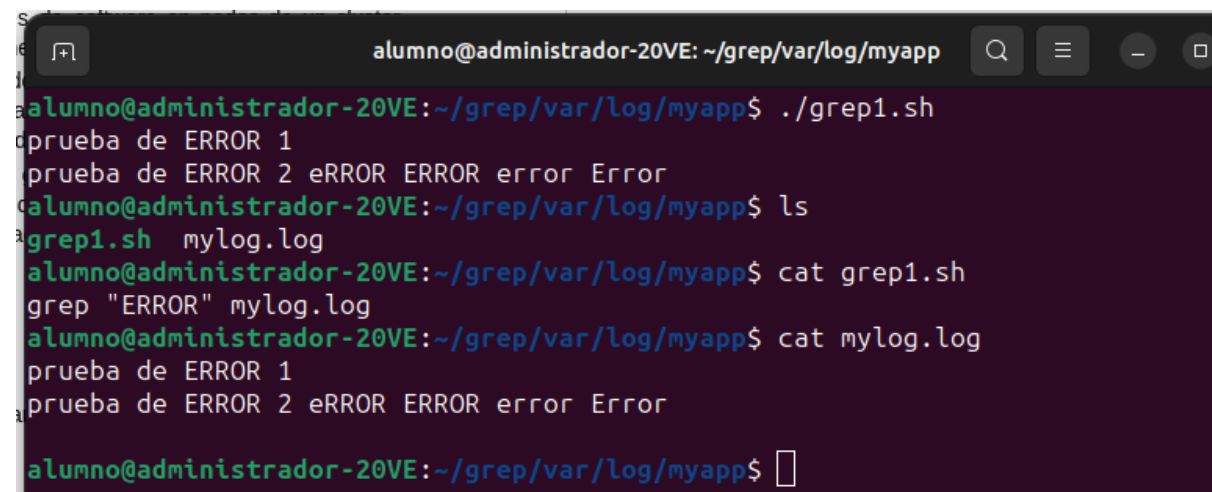
El comando **grep** es una herramienta de línea de comandos disponible en sistemas Unix y Linux utilizada para buscar texto dentro de archivos o flujos de datos. El nombre **grep** proviene de "global regular expression print", refiriéndose a su capacidad para filtrar líneas de texto que coinciden con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

Ejercicios

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

1. Filtrar errores específicos en logs de aplicaciones paralelas:

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



```

alumno@administrador-20VE: ~/grep/var/log/myapp
alumno@administrador-20VE:~/grep/var/log/myapp$ ./grep1.sh
prueba de ERROR 1
prueba de ERROR 2 eRROR ERROR error Error
alumno@administrador-20VE:~/grep/var/log/myapp$ ls
grep1.sh  mylog.log
alumno@administrador-20VE:~/grep/var/log/myapp$ cat grep1.sh
grep "ERROR" mylog.log
alumno@administrador-20VE:~/grep/var/log/myapp$ cat mylog.log
prueba de ERROR 1
prueba de ERROR 2 eRROR ERROR error Error
alumno@administrador-20VE:~/grep/var/log/myapp$

```

Por lo tanto, este comando imprimirá todas las líneas en todos los archivos de log de la aplicación en el directorio myapp/ que contienen la palabra "ERROR". Esto puede ser útil para diagnosticar problemas con la aplicación al identificar errores específicos que se han registrado.

2. 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:~/grep/var/logs/myapp$ sudo ./grep2.sh
pdsh@administrador-20VE: nodo3: connect: timed out
pdsh@administrador-20VE: nodo9: connect: timed out
pdsh@administrador-20VE: nodo2: connect: timed out
pdsh@administrador-20VE: nodo10: connect: timed out
pdsh@administrador-20VE: nodo7: connect: timed out
pdsh@administrador-20VE: nodo4: connect: timed out
pdsh@administrador-20VE: nodo6: connect: timed out
pdsh@administrador-20VE: nodo1: connect: timed out
pdsh@administrador-20VE: nodo5: connect: timed out
pdsh@administrador-20VE: nodo8: connect: timed out

```

Este comando imprimirá una lista de todos los procesos que contienen la cadena 'my_process' en cada uno de los hosts nodo1 a nodo10. Si no hay tales procesos en un host, no se imprimirá nada para ese host. Si el proceso 'my_process' está en ejecución en un host, se imprimirá la línea correspondiente de la salida de ps aux para ese proceso, que incluirá detalles como el ID del proceso (PID), el uso de la CPU y la memoria, el tiempo de inicio, el tiempo de CPU utilizado y el comando.

3. Contar el número de ocurrencias de condiciones de carrera registradas:

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

```

alumno@administrador-20VE:~/grep/var/log$ nano myapp.log
alumno@administrador-20VE:~/grep/var/log$ chmod +x myapp.log
alumno@administrador-20VE:~/grep/var/log$ nano grep3.sh
alumno@administrador-20VE:~/grep/var/log$ chmod +x grep3.sh
alumno@administrador-20VE:~/grep/var/log$ ./grep3.sh
1
alumno@administrador-20VE:~/grep/var/log$ cat myapp.log

"race condition"
alumno@administrador-20VE:~/grep/var/log$ 

```

Este comando imprimirá el número de líneas en el archivo de log /var/log/myapp.log que contienen la frase "race condition". Esto puede ser útil para diagnosticar problemas con una aplicación al identificar cuántas veces se ha registrado una condición de carrera. Si no hay líneas que contengan la frase "race condition", el comando imprimirá "0".

4. Extraer IPs que han accedido concurrentemente a un recurso:

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

```

alumno@administrador-20VE:~/grep/var/log$ nano webserver.log
alumno@administrador-20VE:~/grep/var/log$ chmod +x webserver.log
alumno@administrador-20VE:~/grep/var/log$ nano grep4.sh
alumno@administrador-20VE:~/grep/var/log$ chmod +x grep4.sh
alumno@administrador-20VE:~/grep/var/log$ ./grep4.sh
accessed
alumno@administrador-20VE:~/grep/var/log$ cat webserver.log
accessed resource
accessed resource
alumno@administrador-20VE:~/grep/var/log$ 

```

Este comando imprimirá una lista de direcciones IP únicas que han accedido a un recurso en el servidor web.

5. 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
```

Este comando automatiza la alerta de sobrecarga de memoria en un servicio distribuido enviando un correo electrónico al administrador cuando se detecta una condición de “out of memory”.

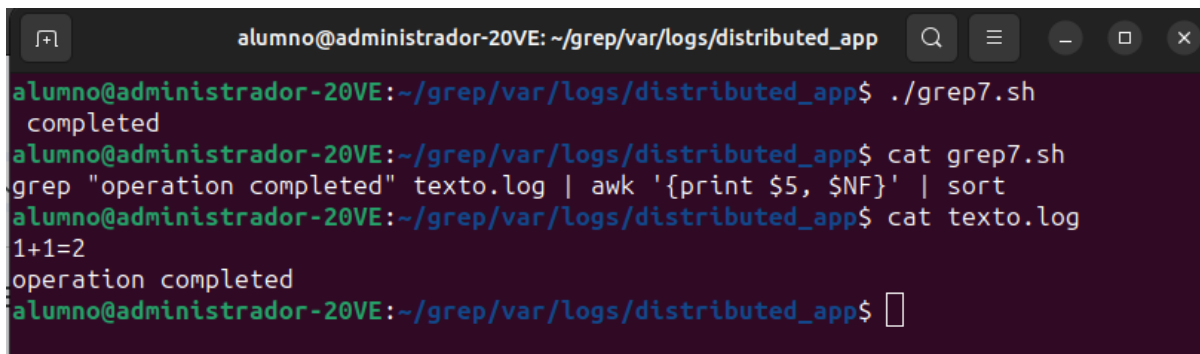
6. Monitorear errores de conexión en aplicaciones concurrentes:

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

Este comando monitorea errores de conexión en aplicaciones concurrentes al buscar la frase “connection error” en el archivo de log de errores de la aplicación y enviar un correo electrónico al administrador con cualquier línea que contenga esa frase.

7. Validar la correcta sincronización en operaciones distribuidas:

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

A terminal window titled 'alumno@administrador-20VE: ~/grep/var/logs/distributed_app' shows the execution of a script. The user runs './grep7.sh', which outputs 'completed'. Then, the user runs 'cat grep7.sh', showing the command 'grep "operation completed" texto.log | awk '{print \$5, \$NF}' | sort'. Next, the user runs 'cat texto.log', which outputs '1+1=2' and 'operation completed'. Finally, the user runs 'alumno@administrador-20VE:~/grep/var/logs/distributed_app\$' and the prompt returns.

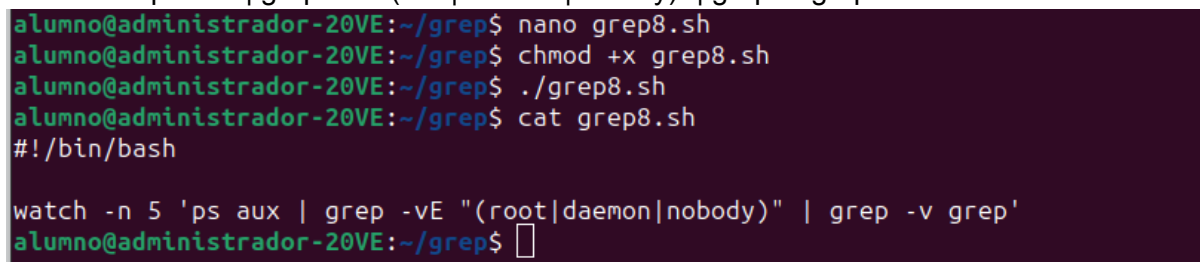
```
alumno@administrador-20VE: ~/grep/var/logs/distributed_app
alumno@administrador-20VE:~/grep/var/logs/distributed_app$ ./grep7.sh
completed
alumno@administrador-20VE:~/grep/var/logs/distributed_app$ cat grep7.sh
grep "operation completed" texto.log | awk '{print $5, $NF}' | sort
alumno@administrador-20VE:~/grep/var/logs/distributed_app$ cat texto.log
1+1=2
operation completed
alumno@administrador-20VE:~/grep/var/logs/distributed_app$
```

Este comando puede ayudar a validar la correcta sincronización en operaciones distribuidas al ordenar las operaciones completadas según los campos seleccionados.

8. Monitorizar la creación de procesos no autorizados

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

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

A terminal window titled 'alumno@administrador-20VE: ~/grep' shows the setup of a script. The user runs 'nano grep8.sh', then 'chmod +x grep8.sh', and finally './grep8.sh'. The script outputs '#!/bin/bash' and the command 'watch -n 5 'ps aux | grep -vE "(root|daemon|nobody)" | grep -v grep''. The user then runs 'cat grep8.sh' and the command is displayed. Finally, the user runs 'alumno@administrador-20VE:~/grep\$' and the prompt returns.

```
alumno@administrador-20VE:~/grep$ nano grep8.sh
alumno@administrador-20VE:~/grep$ chmod +x grep8.sh
alumno@administrador-20VE:~/grep$ ./grep8.sh
#!/bin/bash
watch -n 5 'ps aux | grep -vE "(root|daemon|nobody)" | grep -v grep'
alumno@administrador-20VE:~/grep$ cat grep8.sh
#!/bin/bash
watch -n 5 'ps aux | grep -vE "(root|daemon|nobody)" | grep -v grep'
alumno@administrador-20VE:~/grep$
```



```
alumno@administrador-20VE: ~/grep
Cada 5,0s: ps aux | grep -vE "(... administrador-20VE: Mon Apr 15 14:06:53 2024
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
systemd+      548  0.1  0.0  16204  7168 ?        Ss   11:01   0:11 /lib/systemd/
systemd+      549  0.0  0.0  20776 12544 ?        Ss   11:01   0:04 /lib/systemd/
systemd+      550  0.0  0.0  89692  7296 ?        Ssl  11:01   0:00 /lib/systemd/
nvidia-+      647  0.0  0.0   5152  1920 ?        Ss   11:01   0:00 /usr/bin/nvid
rtkit         962  0.0  0.0  22788  3072 ?        SNsl 11:01   0:00 /usr/libexec/
colord       1338  0.0  0.0 316560 13100 ?        Ssl  11:01   0:00 /usr/libexec/
cups-br+     1604  0.0  0.1 260424 17664 ?        Ssl  11:01   0:00 /usr/sbin/cup
kernoops     1608  0.0  0.0  12520  2056 ?        Ss   11:01   0:00 /usr/sbin/ker
kernoops     1611  0.0  0.0  12520  2048 ?        Ss   11:01   0:00 /usr/sbin/ker
alumno       1809  0.0  0.0  19712 11648 ?        Ss   11:01   0:00 /lib/systemd/
alumno       1810  0.0  0.0 170396  6720 ?        S    11:01   0:00 (sd-pam)
alumno       1816  0.0  0.1  66084 16828 ?        S<sl 11:01   0:02 /usr/bin/pipe
alumno       1819  0.0  0.0 1449640 10976 ?       Ssl  11:01   0:00 /usr/bin/ubun
alumno       1828  0.0  0.1 329604 17280 ?       S<sl 11:01   0:01 /usr/bin/wire
alumno       1832  0.0  0.1  48076 27460 ?       S<sl 11:01   0:01 /usr/bin/pipe
alumno       1834  0.0  0.0 313628  9984 ?       SLsl 11:01   0:00 /usr/bin/gnom
alumno       1835  0.0  0.0  10844  6016 ?        Ss   11:01   0:02 /usr/bin/dbus
alumno       1860  0.0  0.0 608892  7424 ?       Ssl  11:01   0:00 /usr/libexec/
alumno       1864  0.0  0.0 307088  6144 ?       Ssl  11:01   0:00 /usr/libexec/
alumno       1907  0.0  0.0 233252  6016 tty2    Ssl+ 11:01   0:00 /usr/libexec/
alumno       1909  2.9  0.9 26836640 152288 tty2    Sl+  11:01   5:31 /usr/lib/xorg
```

Este comando monitorea la creación de procesos no autorizados al listar todos los procesos que no son ejecutados por los usuarios root, daemon o nobody.

9. 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: ~/grep/var/log
alumno@administrador-20VE:~/grep/var/log$ nano auth.log
alumno@administrador-20VE:~/grep/var/log$ nano grep9.sh
alumno@administrador-20VE:~/grep/var/log$ chmod grep9.sh
chmod: falta un operando después de «grep9.sh»
Pruebe 'chmod --help' para más información.
alumno@administrador-20VE:~/grep/var/log$ chmod +x grep9.sh
alumno@administrador-20VE:~/grep/var/log$ ./grep9.sh
1
alumno@administrador-20VE:~/grep/var/log$ cat grep9.sh
grep "Failed password" auth.log | cut -d' ' -f11 | sort | uniq -c
ead
alumno@administrador-20VE:~/grep/var/log$ cat auth.log
Failed password
Prueba
alumno@administrador-20VE:~/grep/var/log$
```

Este comando detecta y alerta sobre ataques de fuerza bruta SSH al listar las 10 principales direcciones IP desde las que se han producido los intentos de autenticación fallidos.

10. 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      gr  
ep" done
```

```
alumno@administrador-20VE:~/grep$ ./grep10.sh  
alumno@127.0.0.1's password:  
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND  
root           1  0.0  0.0 170084 14272 ?        Ss   11:01   0:04 /sbin/init sp  
lash  
root           2  0.0  0.0      0     0 ?        S    11:01   0:00 [kthreadd]  
root           3  0.0  0.0      0     0 ?        I<   11:01   0:00 [rcu_gp]  
root           4  0.0  0.0      0     0 ?        I<   11:01   0:00 [rcu_par_gp]  
root           5  0.0  0.0      0     0 ?        I<   11:01   0:00 [slub_flushwq  
]  
root           6  0.0  0.0      0     0 ?        I<   11:01   0:00 [netns]  
root           8  0.0  0.0      0     0 ?        I<   11:01   0:00 [kworker/0:0H  
-events_highpri]  
root          10  0.0  0.0      0     0 ?        I<   11:01   0:00 [mm_percpu_wq  
]  
root          11  0.0  0.0      0     0 ?        I    11:01   0:00 [rcu_tasks_kt  
hread]  
root          12  0.0  0.0      0     0 ?        I    11:01   0:00 [rcu_tasks_ru  
de_kthread]  
root          13  0.0  0.0      0     0 ?        I    11:01   0:00 [rcu_tasks_tr
```

```
alumno@administrador-20VE: ~/grep
alumno@172.17.51.159's password:
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0 170084 14272 ?        Ss   11:01   0:04 /sbin/init sp
lash
root         2  0.0  0.0      0     0 ?        S    11:01   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        I<   11:01   0:00 [rcu_gp]
root         4  0.0  0.0      0     0 ?        I<   11:01   0:00 [rcu_par_gp]
root         5  0.0  0.0      0     0 ?        I<   11:01   0:00 [slub_flushwq]
]
root         6  0.0  0.0      0     0 ?        I<   11:01   0:00 [netns]
root         8  0.0  0.0      0     0 ?        I<   11:01   0:00 [kworker/0:0H
-events_highpri]
root        10  0.0  0.0      0     0 ?        I<   11:01   0:00 [mm_percpu_wq]
]
root        11  0.0  0.0      0     0 ?        I    11:01   0:00 [rcu_tasks_kt
hread]
root        12  0.0  0.0      0     0 ?        I    11:01   0:00 [rcu_tasks_ru
de_kthread]
root        13  0.0  0.0      0     0 ?        I    11:01   0:00 [rcu_tasks_tr
ace_kthread]
```

Este comando identifica el uso no autorizado de recursos en clústeres de computación al listar todos los procesos no autorizados que se están ejecutando en cada host.

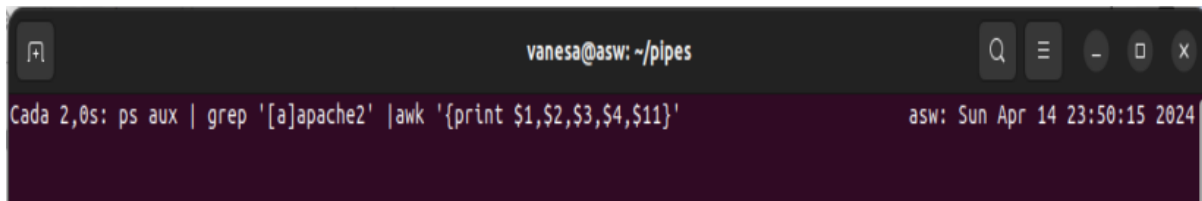
Pipes

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.

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)

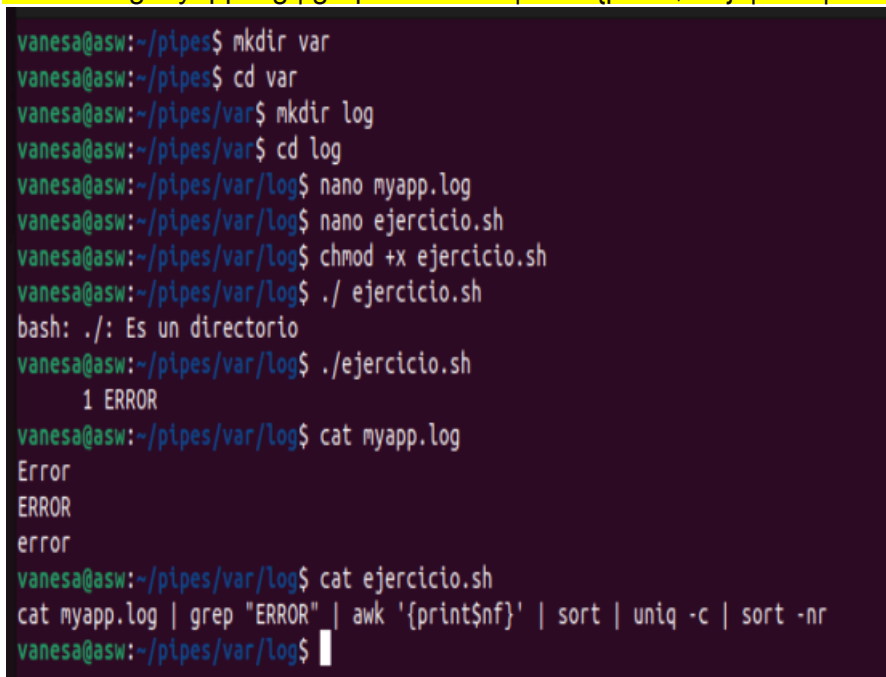
```
watch "ps aux | grep '[a]pache2' | awk '{print \$1, \$2, \$3, \$4, \$11}'"
```



```
vanesa@asw: ~/pipes
Cada 2,0s: ps aux | grep '[a]pache2' | awk '{print $1, $2, $3, $4, $11}'
```

Este comando imprimirá y actualizará continuamente una lista de todos los procesos apache2 en ejecución, mostrando el nombre del usuario, el ID del proceso, el uso de la CPU, el uso de la memoria y el comando para cada proceso. Si no hay procesos apache2 en ejecución, no se imprimirá nada.

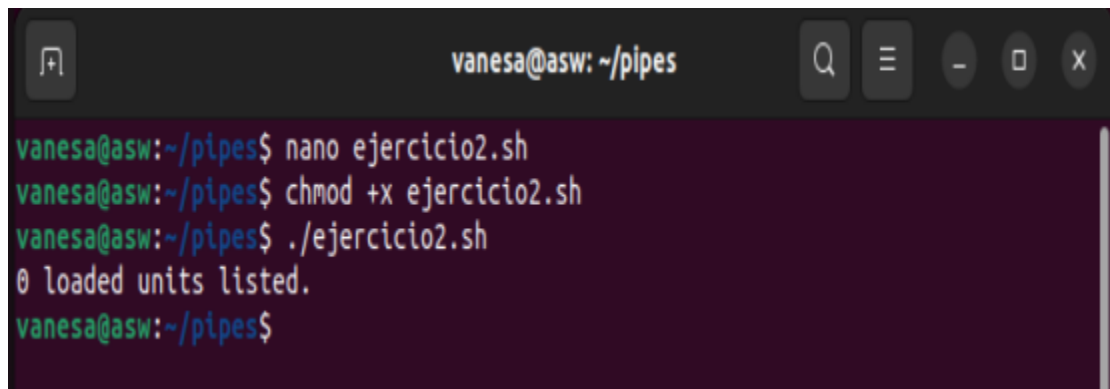
```
cat /var/log/myapp.log | grep "ERROR" | awk '{print $NF}' | sort | uniq -c | sort -nr
```



```
vanesa@asw:~/pipes$ mkdir var
vanesa@asw:~/pipes$ cd var
vanesa@asw:~/pipes/var$ mkdir log
vanesa@asw:~/pipes/var$ cd log
vanesa@asw:~/pipes/var/log$ nano myapp.log
vanesa@asw:~/pipes/var/log$ nano ejercicio.sh
vanesa@asw:~/pipes/var/log$ chmod +x ejercicio.sh
vanesa@asw:~/pipes/var/log$ ./ejercicio.sh
bash: ./: Es un directorio
vanesa@asw:~/pipes/var/log$ ./ejercicio.sh
1 ERROR
vanesa@asw:~/pipes/var/log$ cat myapp.log
Error
ERROR
error
vanesa@asw:~/pipes/var/log$ cat ejercicio.sh
cat myapp.log | grep "ERROR" | awk '{print $NF}' | sort | uniq -c | sort -nr
vanesa@asw:~/pipes/var/log$
```

Este comando imprimirá una lista de los mensajes de error en el archivo de log de la aplicación, ordenados por la frecuencia de ocurrencia, desde el más frecuente hasta el menos frecuente. Si no hay líneas que contengan la palabra "ERROR", el comando no imprimirá nada

```
systemctl --failed | grep "loaded units listed" || systemctl restart $(awk '{print $1}')
```



```
vanesa@asw: ~/pipes
vanesa@asw:~/pipes$ nano ejercicio2.sh
vanesa@asw:~/pipes$ chmod +x ejercicio2.sh
vanesa@asw:~/pipes$ ./ejercicio2.sh
0 loaded units listed.
vanesa@asw:~/pipes$
```

Este comando verifica si hay unidades de systemd que han fallado. Si no hay unidades fallidas, el comando no hace nada. Si hay unidades fallidas, reinicia la primera unidad fallida que se encuentra.

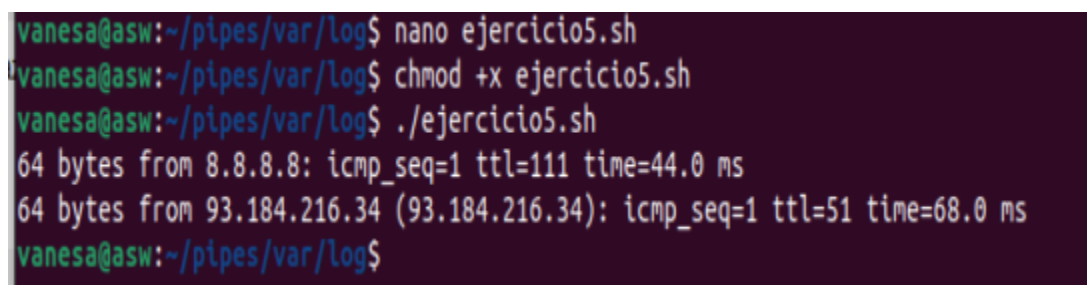
```
3. ps -eo pid,ppid,%cpu,cmd --sort=-%cpu | awk '$3 > 80 {print "Alto uso de CPU: ", $1}' | mail -s "Alerta CPU" admin@example.com
```

Este comando monitorea el uso de la CPU por los procesos en ejecución. Si un proceso está utilizando más del 80% de la CPU, envía un correo electrónico con una alerta al administrador.

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

Este comando busca la cadena 'ERROR' en todos los archivos de log en el directorio /var/log/ y guarda las líneas que contienen 'ERROR' en archivos de texto en un nodo remoto.

```
5. 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
```



```
vanesa@asw:~/pipes/var/log$ nano ejercicio5.sh
vanesa@asw:~/pipes/var/log$ chmod +x ejercicio5.sh
vanesa@asw:~/pipes/var/log$ ./ejercicio5.sh
64 bytes from 8.8.8.8: icmp_seq=1 ttl=111 time=44.0 ms
64 bytes from 93.184.216.34 (93.184.216.34): icmp_seq=1 ttl=51 time=68.0 ms
vanesa@asw:~/pipes/var/log$
```

Este comando realiza un ping a la dirección IP 8.8.8.8 y al dominio www.example.com. Si alguno de los pings falla, registra la fecha y hora del fallo en un archivo llamado fallos_ping.txt.

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



```
vanesa@asw:~/pipes/var/log$ nano ejercicio6.sh
vanesa@asw:~/pipes/var/log$ ./ejercicio6.sh
Apache-CPU: 0 Mem: 0
```

Este comando calcula y muestra el uso promedio de CPU y memoria de todos los procesos httpd (Apache) en ejecución.

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

Este comando verifica si el uso del disco del directorio /home es superior al 80%. Si es así, crea un archivo tar comprimido con gzip del directorio.

8. Monitoreo de uso de CPU por proceso (Python)

```
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")
```

```
alumno@administrador-20VE:~/pipes$ nano pipes8.py
alumno@administrador-20VE:~/pipes$ python3 pipes8.py
alumno@administrador-20VE:~/pipes$
```

- Ejecuta el comando ps para obtener una lista de procesos con su uso de CPU.
- Analiza cada línea de la salida.
- Si el uso de CPU de un proceso supera el 80%, imprime una alerta con el PID y el nombre del proceso.

9. Filtrar líneas de error de un archivo de log (Python)

```
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}")
```

```
vanesa@asw:~/pipes/var/log$ python3 pipes9.py
ERROR: 1
: 1
vanesa@asw:~/pipes/var/log$ cat myapp.log
Error
ERROR
error
vanesa@asw:~/pipes/var/log$
```

- Utiliza grep para filtrar las líneas que contienen la palabra "ERROR" de un archivo de log.
- Cuenta las ocurrencias de cada línea de error.
- Imprime un recuento de cada línea de error

10. Analizar archivo de log grande en paralelo (Python)

```
from multiprocessing import Pool
import subprocess

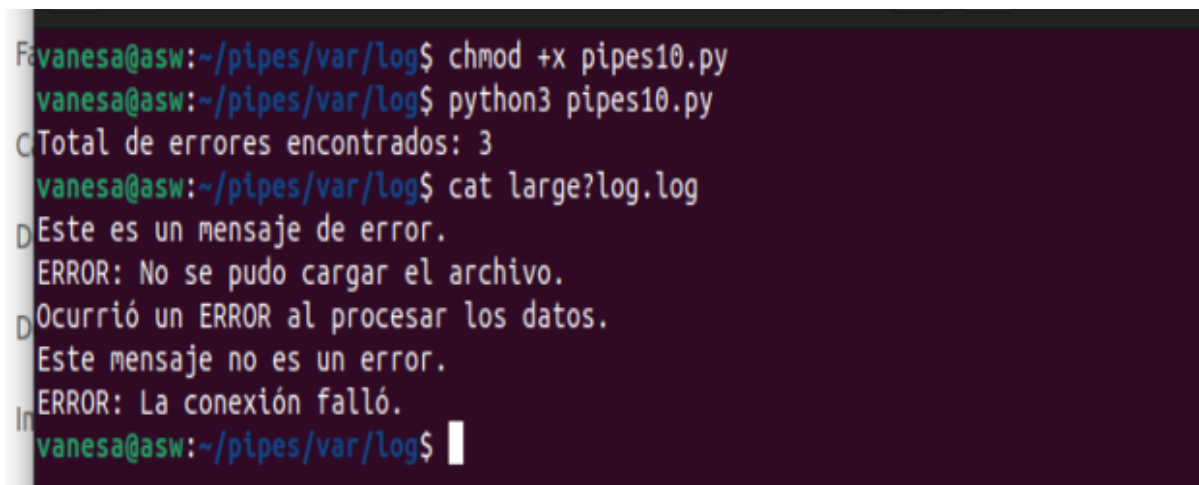
def analyze_log(file_part):
    """Función para analizar una parte del archivo de log."""
    with open(file_part) as f:
        return f.read().count('ERROR')

# Dividir el archivo de log en partes
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))
```



```
Favanesa@asw:~/pipes/var/log$ chmod +x pipes10.py
vanesa@asw:~/pipes/var/log$ python3 pipes10.py
C Total de errores encontrados: 3
vanesa@asw:~/pipes/var/log$ cat large?log.log
D Este es un mensaje de error.
ERROR: No se pudo cargar el archivo.
D Ocurrió un ERROR al procesar los datos.
Este mensaje no es un error.
ERROR: La conexión falló.
In vanesa@asw:~/pipes/var/log$
```

- Divide un archivo de log grande en partes más pequeñas.
- Utiliza multiprocessing para analizar cada parte en paralelo, contando las líneas que contienen "ERROR".
- Imprime el total de errores encontrados en todo el archivo.

11. Monitoreo de cambios en puertos abiertos (Python)

```
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
```

- Ejecuta netstat periódicamente para obtener una lista de puertos abiertos.
- Detecta si hay nuevos puertos abiertos o puertos cerrados desde la última ejecución.
- Imprime los puertos abiertos y cerrados.

12.Cálculo de uso de memoria por usuario (Python)

```
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)
```

```
# Imprimir el uso de memoria por usuario
```

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

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

- Ejecuta ps para obtener el uso de memoria RSS de cada proceso.
- Calcula el uso total de memoria por usuario.
- Imprime el uso total de memoria para cada usuario.

13.Registro de uso de CPU y memoria (Python)

```
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)
```

- Captura el uso de CPU y memoria cada 60 segundos.
- Registra la marca de tiempo, uso de CPU y uso de memoria en un archivo de registro.

14.Monitoreo de uso excesivo de CPU y envío de alerta por correo (Bash)

```
#!/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 los 10 procesos con mayor uso de CPU.
- Si algún proceso supera el 80% de uso de CPU, envía una alerta por correo electrónico.
- Se ejecuta en un bucle indefinido cada 60 segundos.

15. Uso de memoria por usuario (Bash)

```
#!/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."
```

- Ejecuta ps para obtener el uso de memoria RSS de cada proceso.
- Calcula el uso total de memoria por usuario.
- Imprime el uso de memoria por usuario en un archivo.

16. Reporte de uso de CPU y memoria por usuario (Bash)

```
#!/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
```

- Ejecuta ps para obtener el uso de CPU y memoria de cada proceso.
- Agrupa los procesos por usuario y muestra el uso de CPU y memoria para cada usuario.

17. Reporte de memoria para procesos específicos (Bash)

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

- Toma el nombre de un proceso como entrada.
- Ejecuta ps para obtener el uso de memoria de los procesos con ese nombre.
- Imprime el PID, usuario, uso de memoria y comando para cada proceso.
- Calcula y muestra el uso total de memoria para esos procesos.

18.Top IPs en un archivo de log (Bash)

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

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

```
echo "Top IPs"
```

```
awk '{print $1}' $LOG | sort | uniq -c | sort -nr | head -5 | while read COUNT IP; do  
    LOCATION=$(geoipllookup $IP | cut -d, -f2)  
    echo "$IP ($COUNT accesos) - $LOCATION"  
done
```

- Toma un archivo de log como entrada (en este caso, /var/log/httpd/access_log).
- Cuenta las ocurrencias de cada dirección IP en el archivo de log.
- Imprime las 5 direcciones IP con más accesos, junto con su recuento y ubicación geográfica aproximada.

19.Estadísticas de tasa de transferencia de red (Bash)

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

```
NET_DEV="eth0" #aquí cambio
```

```
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
```

- Toma el nombre de un dispositivo de red como entrada (en este caso, eth0).
- Lee los bytes recibidos y transmitidos en un intervalo de 5 segundos.
- Calcula y muestra la tasa de transferencia de recepción y transmisión en bytes/segundo.

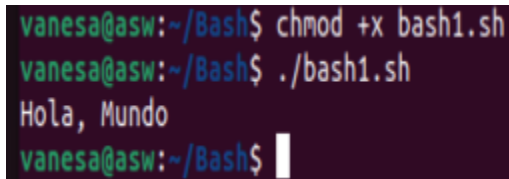
Bash

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>

Variables: Almacenar y manipular datos.

```
nombre="Mundo"  
echo "Hola, $nombre"
```



```
vanesa@asw:~/Bash$ chmod +x bash1.sh  
vanesa@asw:~/Bash$ ./bash1.sh  
Hola, Mundo  
vanesa@asw:~/Bash$
```

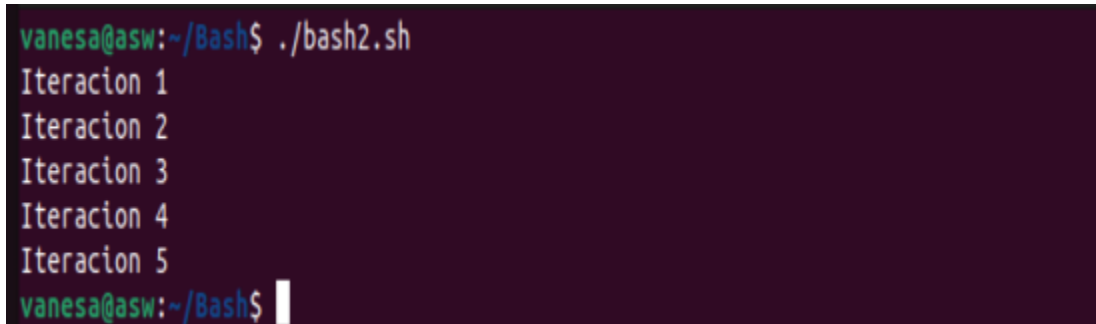
Estructuras de Control: Permiten tomar decisiones y repetir acciones.

```
# If statement  
if [ "$nombre" == "Mundo" ]; then
```

```
    echo "Correcto"  
fi  
# Loop
```

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

```
    echo "Iteración $i"  
done
```



```
vanesa@asw:~/Bash$ ./bash2.sh  
Iteracion 1  
Iteracion 2  
Iteracion 3  
Iteracion 4  
Iteracion 5  
vanesa@asw:~/Bash$
```

Funciones: Agrupar código para reutilizar.

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

```
saludo "Mundo"
```

```

vanesa@asw:~/Bash$ touch bash3.sh
vanesa@asw:~/Bash$ nano bash3.sh
vanesa@asw:~/Bash$ chmod +x bash3.sh
vanesa@asw:~/Bash$ ./bash3.sh
Hola,Mundo
vanesa@asw:~/Bash$

```

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

echo -e "manzana\nbanana\nmanzana" | sort | uniq

```

vanesa@asw:~/Bash$ nano bash4.sh
vanesa@asw:~/Bash$ chmod +x bash4.sh
vanesa@asw:~/Bash$ ./bash4.sh
banana
manzana
vanesa@asw:~/Bash$

```

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

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

```

vanesa@asw:~/Bash$ ls
archivo.txt  bash1.sh  bash2.sh  bash3.sh  bash4.sh  bash5.sh
vanesa@asw:~/Bash$ cat archivo.txt | grep "algo" > resultado.txt
vanesa@asw:~/Bash$ cat resultado.txt
algo interesante
algo más
vanesa@asw:~/Bash$

```

Expresiones regulares: Patrones para buscar y manipular texto.

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

```

vanesa@asw:~/Bash$ nano bash6.sh
vanesa@asw:~/Bash$ chmod +x bash6.sh
vanesa@asw:~/Bash$ ./bash6.sh
404
vanesa@asw:~/Bash$

```

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
```

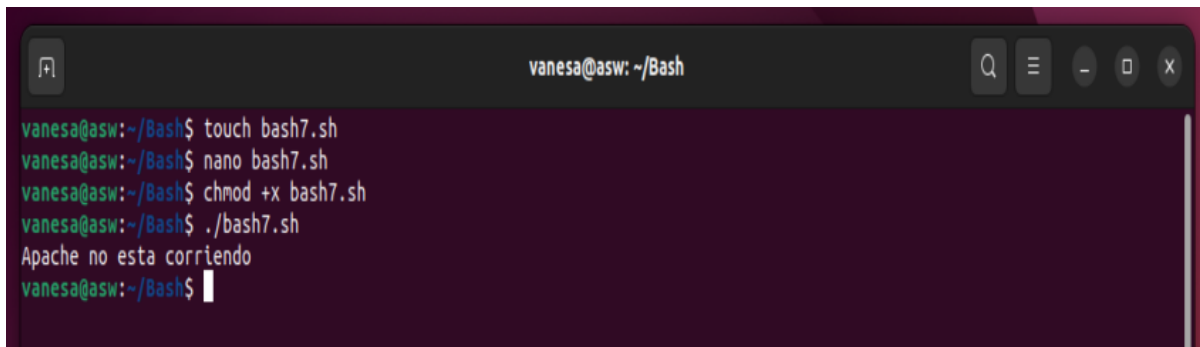
```
if ps aux | grep -q "[a]pache2"; then
```

```
    echo "Apache está corriendo."
```

```
else
```

```
    echo "Apache no está corriendo."
```

```
fi
```

A terminal window titled 'vanesa@asw: ~/Bash' with standard Linux window controls. The user runs a series of commands: 'touch bash7.sh', 'nano bash7.sh', 'chmod +x bash7.sh', and './bash7.sh'. The output of the script is 'Apache no esta corriendo'.

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

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

A terminal window showing the execution of 'cat lista_urls.txt | parallel wget'. It displays the download of a large image from Google Images using wget in parallel. The output includes the URL, connection status, and file size (217K).

```

vanesa@asw:~/Bash$ ls
archivo.txt
bash1.sh
bash2.sh
bash3.sh
bash4.sh
bash5.sh
bash6.sh
bash7.sh
bash8.sh
'ingres?q=imagenes de docker&imgurl=https:%2F%2Flearn.microsoft.com%2Fes-es%2Fdotnet%2Farchitecture%2Fmicroservices%2Fco
ntainer-docker-introduction%2Fmedia%2Fdocker-containers-images-registries%2Ftaxonomy-of-docker-terms-and-concepts.png'
lista_urls.txt
resultado.txt
vanesa@asw:~/Bash$

```

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.

```

vanesa@asw: ~/Bash
vanesa@asw:~/Bash$ nano bash8.sh
vanesa@asw:~/Bash$ chmod +x bash8.sh
vanesa@asw:~/Bash$ ./bash8.sh
Introducetu nombre:Vanesa
Hola,Vanesa
vanesa@asw:~/Bash$

```

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

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

```

vanesa@asw: ~/Bash
vanesa@asw:~/Bash$ find -name "*.txt" | xargs grep "patron"
vanesa@asw:~/Bash$

```

No encuentra ningún patrón.

Ejercicios

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

1. Monitoreo de procesos y terminación de procesos con alto uso de recursos:

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

```
# Configuración
```



```

UMBRALES_CPU=70.0 # Uso máximo de CPU permitido (%)
UMBRALES_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 > $UMBRALES_CPU" | bc -l) )) || [ "$MEM_MB" -gt "$UMBRALES_MEM" ]; then
                echo "$(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

```

Este script monitorea continuamente el uso de CPU y memoria de los procesos especificados en PROCESOS_PARALELOS. Si algún proceso excede los umbrales configurados (UMBRALES_CPU y UMBRALES_MEM), el script registra un mensaje en el archivo de registro (LOG_FILE) y termina el proceso. Además, envía un correo electrónico al administrador (EMAIL_ADMIN) informando sobre el proceso terminado.

2. Respaldo de directorios:

```

#!/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[@]}"

```

```
on directory '/mnt/backup' does not exist. Let's create it using the
alumno@administrador-20VE: ~/bash_ejercicios
alumno@administrador-20VE:~/bash_ejercicios$ mkdir -p /mnt/backup
mkdir: no se puede crear el directorio «/mnt/backup»: Permiso denegado
alumno@administrador-20VE:~/bash_ejercicios$ sudo mkdir -p /mnt/backup
[sudo] contraseña para alumno:
alumno@administrador-20VE:~/bash_ejercicios$ ls
dir1 dir2 dir3 ejercicio2.sh
alumno@administrador-20VE:~/bash_ejercicios$ sudo mkdir -p /mnt/backup
alumno@administrador-20VE:~/bash_ejercicios$ ls
dir1 dir2 dir3 ejercicio2.sh
alumno@administrador-20VE:~/bash_ejercicios$ sudo chown -R alumno:alumno /mnt/backup
alumno@administrador-20VE:~/bash_ejercicios$ ls
dir1 dir2 dir3 ejercicio2.sh
alumno@administrador-20VE:~/bash_ejercicios$ ./ejercicio2.sh
Backup completado para dir1
Backup completado para dir2
Backup completado para dir3
alumno@administrador-20VE:~/bash_ejercicios$
```

Este script realiza un respaldo de los directorios especificados en DIRECTORIOS utilizando el comando tar. Los archivos comprimidos de respaldo se crean en el directorio DESTINO_BACKUP con un nombre que incluye el nombre del directorio y la fecha actual. La función backup_dir es exportada y luego se utiliza parallel para ejecutar la función en paralelo para cada directorio

3. Distribución de tareas en nodos:

```
#!/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
```

```

alumno@administrador-20VE:~/bash_ejercicios$ ./ejercicio3.sh
Asignando tarea1.sh a nodo1
ssh: connect to host nodo1 port 22: Connection timed out
ssh: connect to host nodo2 port 22: Connection timed out
ssh: connect to host nodo3 port 22: Connection timed out
ssh: connect to host nodo1 port 22: Connection timed out
scp: Connection closed
Asignando tarea2.sh a nodo2
ssh: connect to host nodo1 port 22: Connection timed out
ssh: connect to host nodo2 port 22: Connection timed out
scp: Connection closed
Asignando tarea3.sh a nodo3
ssh: connect to host nodo3 port 22: Connection timed out
ssh: connect to host nodo2 port 22: Connection timed out
scp: Connection closed
ssh: connect to host nodo3 port 22: Connection timed out

```

Este script distribuye un conjunto de tareas (scripts de Bash) en varios nodos. Primero, copia cada tarea en el directorio /tmp del nodo correspondiente utilizando scp. Luego, ejecuta cada tarea en el nodo correspondiente mediante ssh. Las tareas se distribuyen de forma circular entre los nodos disponibles.

4. Bloqueo y acceso a un recurso compartido:

```
#!/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
```

```

alumno@administrador-20VE:~/bash_ejercicios$ nano ejercicio4.sh
alumno@administrador-20VE:~/bash_ejercicios$ chmod +x ejercicio4.sh
alumno@administrador-20VE:~/bash_ejercicios$ ./ejercicio4.sh
Accediendo al recurso

```

Este script implementa un mecanismo de bloqueo para acceder a un recurso compartido. La función `adquirir_lock` intenta crear un archivo de bloqueo (`LOCK_FILE`) de forma exclusiva. Si el archivo ya existe, el script esperará hasta poder crear el archivo. Una vez que se adquiere el bloqueo, el script puede acceder y trabajar con el recurso compartido (`RECURSO`). Finalmente, la función `liberar_lock` elimina el archivo de bloqueo para permitir que otros procesos accedan al recurso.

5. Recolección de métricas de nodos:

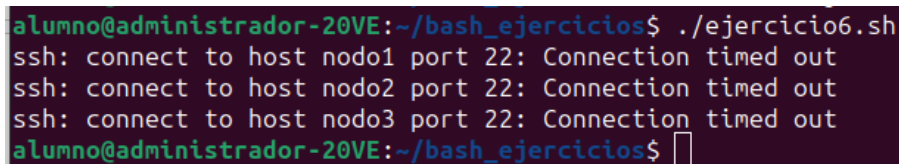
```
#!/bin/bash
```

```
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/.*,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
```



```
alumno@administrador-20VE:~/bash_ejercicios$ ./ejercicio6.sh  
ssh: connect to host nodo1 port 22: Connection timed out  
ssh: connect to host nodo2 port 22: Connection timed out  
ssh: connect to host nodo3 port 22: Connection timed out  
alumno@administrador-20VE:~/bash_ejercicios$
```

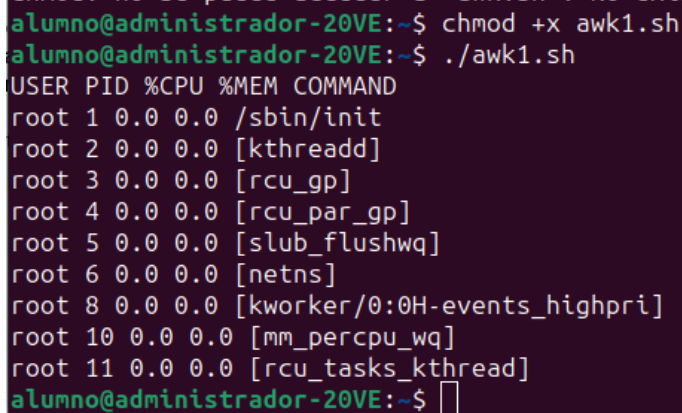
Este script recolecta métricas de uso de CPU, memoria y disco de varios nodos y las almacena en un archivo CSV (ARCHIVO_METRICAS). Se conecta a cada nodo mediante ssh y ejecuta comandos para obtener las métricas deseadas. Luego, agrega una línea con el nombre del nodo y las métricas correspondientes al archivo CSV.

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.

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
```



```
alumno@administrador-20VE:~$ chmod +x awk1.sh
alumno@administrador-20VE:~$ ./awk1.sh
USER PID %CPU %MEM COMMAND
root 1 0.0 0.0 /sbin/init
root 2 0.0 0.0 [kthreadd]
root 3 0.0 0.0 [rcu_gp]
root 4 0.0 0.0 [rcu_par_gp]
root 5 0.0 0.0 [slub_flushwq]
root 6 0.0 0.0 [netns]
root 8 0.0 0.0 [kworker/0:0H-events_highpri]
root 10 0.0 0.0 [mm_percpu_wq]
root 11 0.0 0.0 [rcu_tasks_kthread]
alumno@administrador-20VE:~$
```

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

Este comando muestra los primeros 10 procesos del sistema, mostrando sólo los campos de usuario (usuario que ejecuta el proceso), PID (ID del proceso), %CPU (uso de CPU), %MEM (uso de memoria) y la línea de comando del proceso.

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>

Este script divide un archivo de entrada grande (input.txt) en múltiples archivos de salida más pequeños (output0.txt, output1.txt, output2.txt, etc.). Cada archivo de salida contendrá 1000 líneas del archivo de entrada.

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}'
```

Este comando monitorea un archivo de registro (/var/log/app.log) en tiempo real y filtra las líneas que contienen la palabra "ERROR". Luego, utilizando awk, imprime el primer campo (\$1), el segundo campo (\$2) y el último campo (\$NF) de cada línea filtrada.

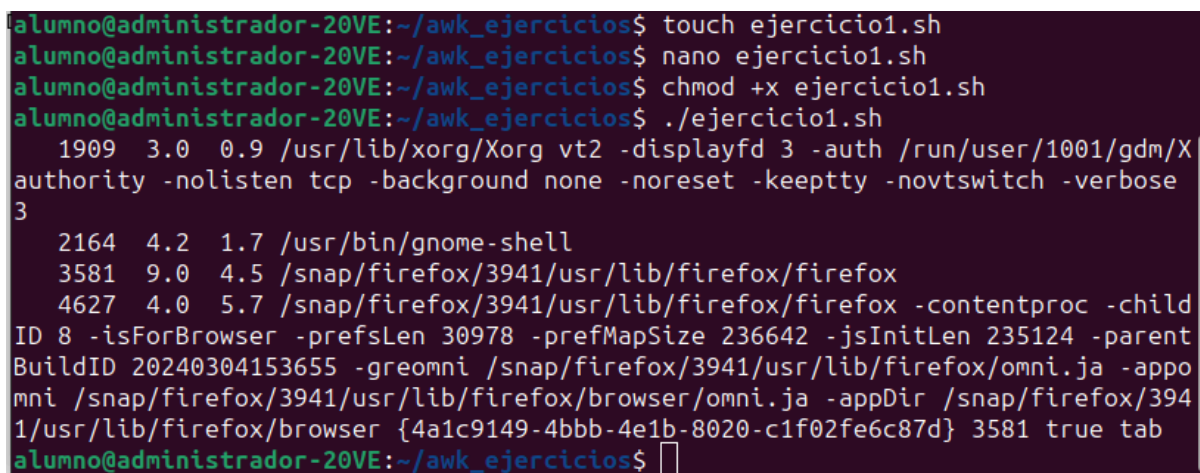
```
ps -eo user,pid,pcpu,pmem,cmd | grep apache2 | awk '$3 > 50.0 || $4 > 50.0 {print "Alto recurso: ", $0}'
```

Este comando muestra los procesos de Apache2 que tienen un uso de CPU superior al 50% o un uso de memoria superior al 50%.

Ejercicios:

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

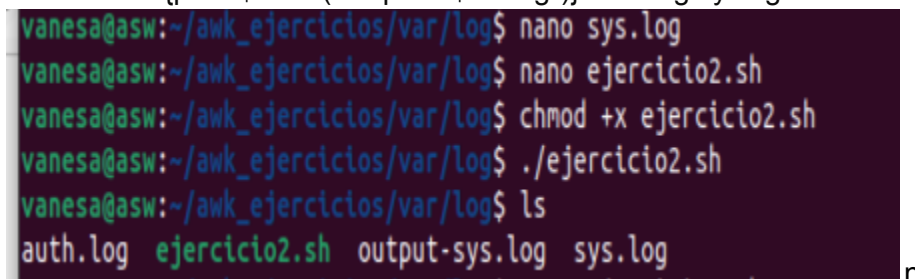
1. `ps -eo pid,pcpu,pmem,cmd | awk '$2 > 10.0 || $3 > 10.0'`



```
alumno@administrador-20VE:~/awk_ejercicios$ touch ejercicio1.sh
alumno@administrador-20VE:~/awk_ejercicios$ nano ejercicio1.sh
alumno@administrador-20VE:~/awk_ejercicios$ chmod +x ejercicio1.sh
alumno@administrador-20VE:~/awk_ejercicios$ ./ejercicio1.sh
1909  3.0  0.9 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1001/gdm/X
authority -nolisten tcp -background none -noreset -keeptty -novtswitch -verbose
3
2164  4.2  1.7 /usr/bin/gnome-shell
3581  9.0  4.5 /snap/firefox/3941/usr/lib/firefox/firefox
4627  4.0  5.7 /snap/firefox/3941/usr/lib/firefox/firefox -contentproc -child
ID 8 -isForBrowser -prefsLen 30978 -prefMapSize 236642 -jsInitLen 235124 -parent
BuildID 20240304153655 -greomni /snap/firefox/3941/usr/lib/firefox/omni.ja -appo
mni /snap/firefox/3941/usr/lib/firefox/browser/omni.ja -appDir /snap/firefox/394
1/usr/lib/firefox/browser {4a1c9149-4bbb-4e1b-8020-c1f02fe6c87d} 3581 true tab
alumno@administrador-20VE:~/awk_ejercicios$
```

Este comando muestra los procesos que tienen un uso de CPU mayor al 10% o un uso de memoria mayor al 10%.

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



```
vanesa@asw:~/awk_ejercicios/var/log$ nano sys.log
vanesa@asw:~/awk_ejercicios/var/log$ nano ejercicio2.sh
vanesa@asw:~/awk_ejercicios/var/log$ chmod +x ejercicio2.sh
vanesa@asw:~/awk_ejercicios/var/log$ ./ejercicio2.sh
vanesa@asw:~/awk_ejercicios/var/log$ ls
auth.log  ejercicio2.sh  output-sys.log  sys.log
```

Este script divide las líneas del archivo /var/log/syslog en múltiples archivos de registro (output-\$4.log) basados en el cuarto campo (\$4) de cada línea.

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

```

vanesa@asw:~/awk_ejercicios/var/log$ nano auth.log
vanesa@asw:~/awk_ejercicios/var/log$ nano auth.log
vanesa@asw:~/awk_ejercicios/var/log$ nano ejercicio3.sh
vanesa@asw:~/awk_ejercicios/var/log$ chmod +x ejercicio3.sh
vanesa@asw:~/awk_ejercicios/var/log$ ./ejercicio3.sh
    2 Failed password
vanesa@asw:~/awk_ejercicios/var/log$ cat auth.log
Failed password
Failed password
vanesa@asw:~/awk_ejercicios/var/log$

```

Este comando analiza el archivo /var/log/auth.log y cuenta el número de intentos fallidos de inicio de sesión por dirección IP.

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

Nuevo archivo creado: archivo1.txt

Nuevo archivo creado: archivo2.pdf

Nuevo archivo creado: archivo3.jpg

Este script utiliza inotifywait para monitorear un directorio (/path/to/dir) y detectar cuando se crea un nuevo archivo. Luego, a través de un pipe (|), envía la salida a awk, que imprime la cadena "Nuevo archivo creado:" seguida del nombre del archivo recién creado (\$3).

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

```

vanesa@asw:~/awk_ejercicios$ ./ejercicio5.sh
Espacio total usado por archivos .py:
vanesa@asw:~/awk_ejercicios$ ls
ejercicio5.sh  var
vanesa@asw:~/awk_ejercicios$

```

Este script utiliza find para buscar todos los archivos con extensión .py en el directorio actual y subdirectorios. Luego, ejecuta ls -l en esos archivos y canaliza la salida a awk.

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

```

vanesa@asw:~/awk_ejercicios$ nano ejercicio6.sh
vanesa@asw:~/awk_ejercicios$ chmod +x ejercicio6.sh
vanesa@asw:~/awk_ejercicios$ ./ejercicio6.sh
Tiempo promedio de respuesta: 37,3333
vanesa@asw:~/awk_ejercicios$

```

Este script procesa un archivo de registro de acceso (access.log) y calcula el tiempo promedio de respuesta. Asume que el último campo (\$NF) de cada línea es el tiempo de respuesta. awk suma todos los tiempos de respuesta (sum+=\$NF) y, al final (END), divide la suma total por el número de líneas (NR) para obtener el promedio.

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

```
vanesa@asw:~/awk_ejercicios$ ./ejercicio7.sh
Espera (D): - Ejecución (R): 1
vanesa@asw:~/awk_ejercicios$
```

Este script usa `ps -eo state` para obtener el estado de todos los procesos en ejecución. Luego, `awk` analiza la salida y cuenta el número de procesos en estado de espera (`/D/ {d++}`) y en ejecución (`/R/ {r++}`). Al final (`END`), imprime el recuento de cada estado.

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

Este script es idéntico al anterior, pero sin el `*` al final. Imprime el recuento de procesos en espera y 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`

Alerta: Uso excesivo de swap

Este script analiza el archivo `/proc/meminfo` y verifica si el uso de la memoria de intercambio (swap) es excesivo. Busca las líneas que comienzan con "SwapTotal" y "SwapFree", y asigna los valores correspondientes a las variables `total` y `free`. Al final (`END`), calcula el porcentaje de uso de swap como $((total-free)/total*100)$ y, si es mayor al 20%, imprime un mensaje de alerta.

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

```
vanesa@asw:~/awk_ejercicios$ nano ejercicio11.sh
vanesa@asw:~/awk_ejercicios$ chmod +x ejercicio11.sh
vanesa@asw:~/awk_ejercicios$ ./ejercicio11.sh
Uso total de disco (sin subdirectorios): 660
vanesa@asw:~/awk_ejercicios$
```

Este script utiliza `ls -l` para obtener una lista detallada de archivos y directorios en el directorio actual. La salida se canaliza a `awk`. `awk` filtra las líneas que no comienzan con "total" (`!/^total/`) y que no son directorios (`!/^d/`), luego suma el tamaño de cada archivo (`sum += $5`, donde `$5` es el tamaño del archivo en bytes). Al final (`END`), imprime el uso total de disco de los archivos, excluyendo los subdirectorios.