

## Exercício PL3

1) Run ps with -ef option first, and then with -aux options and notice the differences.

PS -ef

-e : mostra todos os processos no sistema

-f : exibe os processos num formato completo, que inclui informações detalhadas como PID, UID, PPID, etc...

lista todos os processos no formato padrão

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	10:00	?	00:00:05	/sbin/limit
user	1234	1	40001	10:01	?	00:00:01	/usr/bim/bi

UID: nome do utilizador que iniciou o processo.

PID: identificação do processo

PPID: o identificador do processo pai (Parent Process ID)

C: Percentagem de uso da CPU.

STIME: Hora de inicio do processo.

TTY: O terminal associado ao processo ou ? se não houver terminal

TIME: Tempo total de CPU consumido.

CMD: Comando utilizado para iniciar o processo

PS -aux

a: mostra processos de todos os utilizadores

u: exibe detalhes como o nome de utilizador e percentagens de CPU/memória

x: inclui processos que não estão ligados a um terminal

lista todos os processos no formato BSD

(Berkeley Software Distribution)	USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START TIME	CMD
root	1	0.0	0.1	16560	1268	?	Ss	10:00	0:05	/sbin/limit

VSZ: tamanho da memória virtual que o processo está a usar (em KB)

RSS: " " " física " " " "

STAT: Estado do processo (S: sleeping ; R: running, etc)

Resumo: ps -ef é mais útil para analisar relações entre processos (PID/PPID). ps-aux é melhor para monitorizar consumo de recursos (CPU e memória).

2) Run ps with the -l option. Filter the output, so that only the process ID, priority, nice value, and the process command line are displayed.

ps -l → exibe informações mais detalhadas do que o comando ps.

F VID PID PPID C PRI NI SZ RSS WCHAN STAT TTY TIME CMD  
1 1000 1234 1233 0 20 0 1056 2048 - S pts/0 00:00 bash

F: flags do processo (um campo interno do sistema).

PRI: Prioridade do processo

NI: Valor nro do processo (determina a prioridade)

SZ: tamanho da memória virtual usada (em páginas)

RSS: memória residente (em KB)

WCHAN: função em que o processo está suspenso (se aplicável)

→ (os números estão mal)

ps -l | awk '{print \$3, \$6, \$7, \$14}' | column -t

↳ para organizar por coluna

VID C PRI PID PRI NI CMD

3) Run the top command and watch the output as it changes.

top: ferramenta interativa para monitorar processos ativos e o desempenho do sistema em tempo real. A saída é constantemente atualizada, mostrando mudanças no consumo de recursos e no estado dos processos.

Parte superior ↗

Uptime: indica há quanto tempo o sistema está ativo

Tasks: n: total de processos (tasks), incluindo os estados (running, sleeping, stopped, zombie).

CPU usage: porcentagem de uso da CPU (usuário, sistema, inativo, etc...)

Memory usage: estatísticas de memória física (RAM) e SWAP.

Parte inferior ↗

PID

USER

%CPU

%MEM

TIME+: tempo total da CPU consumido

COMMAND

→ enquanto o top estiver a ser executado, podemos interagir com ele :

q: sai do top

h: mostra a ajuda (atâlhos disponíveis)

K: termina um processo (será solicitado o PID)

R: altera o valor de nice de um processo (será solicitado o PID e um novo valor)

P: ordena os processos pelo uso de CPU

M: ordena os processos pelo uso de memória

T: ordena os processos pelo tempo de execução

SPACE: atualiza imediatamente a saída

1: mostra estatísticas detalhadas para cada CPU

→ estes comandos podem ser usados enquanto o top está a rodar!

ao executar o top, notarás :

→ flutuações no uso da CPU e RAM

→ processos em destaque - aqueles que consumem mais recursos serão estilizados na topo da lista

4) Start several processes in the background executing the sleep command with different sleep times.

sleep 300 &

sleep 300 &

sleep 600 &

sleep 700 &

a. get their PIDs with the ps command

ps -ef | grep sleep (ou simplesmente ps)

5012

5013

5014

5015

b. Send the SIGSTOP and SIGCONT signals to some of them, with the kill command. Use the ps command to confirm the change

Kill -SIGSTOP [PID]

→ suspende o processo

Kill -SIGCONT [PID]

→ retorna um processo que foi suspenso

→ Para afiar o estado do processo : ps -eo pid,stat,command | grep sleep

c. Terminate some processes with the SIGTERM signal and the remaining ones with the SIGKILL signal.

(-15)

Kill -SIGTERM [PID] → realiza uma saída limpa

Kill -SIGKILL [PID], → força o término do processo  
(-9)

o processo terminado com SIGKILL  
não aparece na lista (ps), é  
removido do sistema imediatamente.

(5) The stress command is a utility program that will put stress on your system's CPU, RAM, I/O, etc..

↳ O comando stress é uma ferramenta usada para gerar carga num sistema, testar a resistência de componentes como a CPU, RAM, I/O...

É útil para testar a estabilidade do sistema sob alta carga ou para avaliar o desempenho do hardware em situações extremas.

a. Install it with "sudo apt install stress".

b. Run it with "stress cpu --2".

c. Press control + Z to pause the program and give us control of the terminal again.

d. Use the ps -u command to see the current state of the process.

[ps -u] : usado para listar os processos de um usuário específico. Ao seu lado, usado para fornecer detalhes de todos os processos em execução pelo meu e à frente, utilizador, juntamente com detalhes como o PID, o tempo de o meu nome CPU utilizado, o estado do processo e o comando que está a de utilizador ser executado.

(lines) R: running S: sleeping T: stopped (mudando apos (CTRL+Z))  
Z: zombie I: idle

e. Use the jobs command to see the current state of the process.

→ O comando jobs é utilizado para listar os trabalhos que estão a correr em segundo plano ou em suspenso no terminal atual. Este comando é útil para ver o estado de processos que foram enviados para o fundo com o uso de & ou suspensos com CTRL+Z.

f. To bring it back to the foreground, you can use the fg command, or if you want the program to run in the background, you can just use the bg command.

b. Change it to allow the user to specify the number of seconds

```
#!/bin/bash  
#barra "a script vai dormir agora por $1 segundos"  
sleep $1  
echo "já dormi por $1 segundos"  
    ↗ se quisermos 5 segundos
```

↳ us testar e só põi \$ ./my\_suit.sh 5

⑦ Add the following new functionality to the guessNumberGame script: for each user, the best score he/she has ever achieved should be stored persistently. When each round ends, the user must be informed if he/she has or not beaten his/her previous best score. Tips:  
a. To be persistent, the best score must be stored in a file. To be personal, it should be stored in the user's home folder, for instance ~/guessNumberGame.topScore.

b. To get the file's content into a variable use VARNAME=\$(cat ~/guessNumberGame.topScore). Remember that on the first round the file will not exist.

c. To place a new value in the file just echo the variable's value redirecting the output to the file

```
#!/bin/bash  
TOP_SCORE_FILE="$HOME/.guessNumberGame.topScore"  
echo "I've a number between 0 and 100, it may even be 0 or 100."  
echo "Try to guess it!"  
num=$(shuf -i 0-100 -n 1) → gerar um n: aleatório  
trials=0  
guess=101  
if [ -f "$TOP_SCORE_FILE" ]; then  
    BEST_SCORE=$(cat "$TOP_SCORE_FILE") → li a pontuação anterior  
    no arquivo, se  
    existir  
else  
    BEST_SCORE=0  
fi
```

(fg)  
Em foreground → o comando ocupa o terminal e o shell não retorna  
o prompt até o comando terminar

(bg)  
Em background → o comando é executado em segundo plano (com &) e o terminal  
retorna imediatamente o prompt.

Usar jobs → para ver os trabalhos em execução ou suspensos no background

Usar ps → para verificar o estado e o terminal do processo

Usar top → para monitorizar processos em tempo real.

g. With the stress command running in the background, use the top command.

↳ Saber os que afetam primeiro porque têm grande carga  
de CPU (1.49 / 1.50)

h. Terminate all the stress processes with the kill command. Alternatively,  
you can use the killall command to terminate all the stress commands  
by name.

Killall stress → termina todos os processos stress de uma vez

Kill -9 [PID] → termina um processo de cada vez

↳ depois de enviar o sinal de término, posso verificar se os processos  
foram efetivamente terminados utilizando novamente o comando ps ou  
#ps aux | grep stress

⑥ The following script sleeps for 10 seconds

a. Create the script file with the shown content and test it.

nano meu-script.sh → Criar o arquivo

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

```
sleep 10
```

```
echo "I've been sleeping for 10 seconds. I want more!"
```

ctrl O e ctrl N → salvar e sair

chmod u+x meu-script.sh → tornar o script executável

While [ "\$num" != "\$guess" ]; do  
    Read -p "Enter your guess please: " guess  
    → Loop para o utilizador  
        adivinhão o numero

```
if ! [[ "$guess" =~ ^[0-9]+\$ ]]; then  
    echo "Please enter a valid number!"  
    continue  
fi
```

$$\text{tries} = \$((\text{tries} + 1))$$

if [ "\$gues" -gt "\$num" ]; then

echo "Sorry, too large!"

```
elif L["guess"] - lt "3 num  
echo "Sorry, too small!"
```

pi  
done

echo "Very well, you have guessed it in \$tries tries."

```
if [ "$Best SCORE" -eq 0 ] || [ "$tries" -lt "$BEST SCORE" ];  
echo "Congratulations! You've set a new best score with $tries tries!"  
echo "$tries" > "$TOP SCORE-FILE"
```

else

echo "you did not beat your best score. your best score is \"\$BEST SCORE\"";

10

⑧ Develop a script that repeatedly prompts the user to enter positive integers and calculates the largest, smallest, and average of the entered numbers.

The script should continue to accept numbers until the user enters a value of zero which will end the input sequence. Afterward, the script should display the largest number, smallest number, and the average of all entered numbers (excluding the zero).

↳ irá a folha

#!/bin/bash

echo "Enter positive integers one by one. Enter 0 to stop."

largest=0  
smallest=0  
sum=0  
count=0

} initialization of Variables

while true; do

read -p "Enter a number:" num

if ! [[ "\$num" =~ ^[0-9]+\\$ ]]; then  
echo "Please enter a valid positive integer"  
continue

fi

if [ "\$num" -eq 0 ]; then  
break  
fi

count=\$((count + 1))  
sum=\$((sum + num))

→ implements the counter and the sum

if [ "\$count" -eq 1 ]; then

largest=\$num

smallest=\$num

else

if [ "\$num" -gt "\$largest" ]; then

largest=\$num

fi

→ updates the maximum value

or minimum value

if [ "\$num" -lt "\$smallest" ]; then

smallest=\$num

fi

done

```
if [ "$count" -gt 0 ]; then  
    average=$(echo "scale=2; $sum / $count" | bc)
```

else

average = 0

fi

calcular a media

```
echo "Largest number:" $largest  
echo "Smallest number:" $smallest  
echo "Average:" $average
```

Q) The following script lists every valid user's login name and home folder. The output of the getent command is sent (through a pipe) to the cut command to extract required data. We want fields 1 (username) and 6 (home folder). For each user, by using variable expansion with suffix and prefix pattern removal, the two elements are isolated.

a. Create the script with the shown content and test it.

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

```
userlist=$(getent passwd | cut -d ":" -f 1,6)
```

```
for user in $userlist; do
```

```
    user=${user%:*}
```

```
    home=${user#*:}
```

```
    echo "$user - $home"
```

done