Taxa de Escrita e Leitura de Processos em bash

Sistemas Operativos

Departamento de Eletrónica,

Telecomunicações e Informática

Trabalho 1 – Turma P5

Filipe Obrist – 107471

Rodrigo Aguiar – 108969

2022/2023

Índice

Introdução

No âmbito da disciplina de Sistemas Operativos, foi proposta a execução de um script em bash que permitisse obter dados sobre os processos, nomeadamente a sua taxa de leitura e de escrita em Bytes.

Estes dados serão mostrados ao utilizador através de uma tabela, que poderá ser alterada conforme as opções de ordenação escolhidas pelo utilizador (-r para inverter a ordem, -w para ordenar pelos valores de escrita e -p para visualizar um número específico de processos) .

Também será possível ao utilizador procurar apenas processos que pretende segundo um conjunto de regras, pelo nome (opção -c) , utilizador (opção -u), por data de inicio e fim (opções -s e -e) e pelo id do processo (-m e -M).

Neste documento, serão demonstrados e explicados os métodos utilizados para a realização deste script, bem como vários testes e gerenciamento das várias opções e argumentos permitidos na execução do mesmo.

Metodologia

Variáveis e estruturas de dados

```
declare -A options=()  # associative array where the options/arguments will be stored
declare -A Rbytes=()   # associative array where Rbytes will be stored for each PID
declare -A Wbytes=()   # associative array where Wbytes will be stored for each PID
declare -A processinfo=() # associative array where the info of each process will be stored
i=0 # variable used to exit out of program if wrong arguments are given
regexnumber=^[0-9]*$                                    # regex for numbers
regexletras=^[a-zA-Z]+$                                 # regex for letters
```

Figura 1 - Variáveis e Estruturas de Dados

De forma a facilitar o armazenamento de informação sobre os processos e as opções permitidas no script, foram criados vários arrays associativos, que permitiram o armazenamento de informação sobre cada processo.

Também foram criados vários formatos de Regex e uma variável i (inicializada a 0), para garantir que a passagem de argumentos ao script era feita conforme o formato pretendido, saindo do script quando tal não acontecesse.

Obtenção dos Process ID's

Para obter a informação inicial sobre os processos, foi utilizado o comando ps -e, através do qual foi extraída a 1ª coluna de informação, que continha os PID's de todos os processos.

```
processes=$(ps -e)
pids=$(echo "$processes" | tail -n +2 | awk '{ print $1}')  # get the pids from the processes
```

Figura 2 - Obtenção dos PID

Posteriormente, os process id's (PID's) foram armazenados num array chamado arraypids. Houve a necessidade de remover o último elemento do array, que representava o PID do processo ps -e, o qual não era pretendido obter informação sobre.

```
mapfile -t arraypids < <(echo "${pids}")
unset 'arraypids[-1]' # the ps -e process itself shouldnt be stored in the array,
```

Figura *3 – Criação do array de PID's*

Função Arguments_guide()

```
function Arguments_guide(){
    echo "+----------+-------------------------------------------------+------------------------------+"
    echo "|  Option  |                  Description                    |             Example          |"
    echo "+----------+-------------------------------------------------+------------------------------+"
    echo "| -c       | Search processes by name ( regular expression)  |       -c bash                |"
    echo "| -s       | Search processes started after a certain date   |       -s \"Sep 20 10:00\"      |"
    echo "| -e       | Search processes started before a certain date  |       -e \"Dec 13 19:24\"      |"
    echo "| -u       | Search processes by user                        |       -u joão                |"
    echo "| -m       | Search processes with a higher PID than the arg |       -m 10                  |"
    echo "| -M       | Search processes with a lower PID than the arg  |       -M 100                 |"
    echo "| -p       | Search a specific number of processes           |       -p 25                  |"
    echo "| -r     | Sort the table by reverse order                |   -r (arguments are ignored) |"
    echo "| -w       | Sort the table by write values                  |   -w (arguments are ignored) |"
    echo "+----------+-------------------------------------------------+------------------------------+"
    echo "|By default the table is sorted by read values decreasingly                                    |"
    echo "|The last argument must be a number -> time to wait between reads of the processes       |"
    echo "+---------------------------------------------------------------------------------------------+"
}
```

Figura 4 – Tabela de ajuda

A função Arguments_guide() serve como um guia, indicando ao utilizador quais as opções e respetivos argumentos que pode utilizar quando executa o script, contendo também vários exemplos para uma compreensão mais fácil.

Esta função pode ser invocada utilizando a opção -h na execução do script.

```
finex@DESKTOP-M0VVTB3:/mnt/c/Users/rodri/OneDrive/Ambiente de Trabalho/Uni/ano2-sem1/SO/Trabalho 1$ bash rwstat.sh -h
+----------+-------------------------------------------------+------------------------------+
|  Option  |                  Description                    |             Example          |
+----------+-------------------------------------------------+------------------------------+
| -c       | Search processes by name ( regular expression)  |       -c bash                |
| -s       | Search processes started after a certain date   |       -s "Sep 20 10:00"      |
| -e       | Search processes started before a certain date  |       -e "Dec 13 19:24"      |
| -u       | Search processes by user                        |       -u joão                |
| -m       | Search processes with a higher PID than the arg |       -m 10                  |
| -M       | Search processes with a lower PID than the arg  |       -M 100                 |
| -p       | Search a specific number of processes           |       -p 25                  |
| -r       | Sort the table by reverse order                 |   -r (arguments are ignored) |
| -w       | Sort the table by write values                  |   -w (arguments are ignored) |
+----------+-------------------------------------------------+------------------------------+
|By default the table is sorted by read values decreasingly                                    |
|The last argument must be a number -> time to wait between reads of the processes       |
+---------------------------------------------------------------------------------------------+
```

Figura 6 – Opção -h

Função Argument_checker()

Uso da função

Esta função é responsável por analisar todos as opções e argumentos fornecidos ao script na sua execução e verificar se estes são os corretos e se apresentam os seus valores conforme o formato pretendido.

Esta função tem por base a função built-in da bash getopts, que permite facilitar bastante o gerenciamento de opções e argumentos no script.

Figura 7 – Uso da função getopts

A função getopts é invocada num while loop, ou seja, irá percorrer todos as opções passadas na execução do script 1 a 1, guardando o seu valor na variável option.

As opções permitidas neste script são:

- -c expressão regular -> procurar processos pelo nome
- -s data -> procurar processos que começaram depois da data dada
- -e data -> procurar processos que começaram antes da data dada
- -u nome -> procurar processos pelo nome de utilizador
- -m PID -> procurar processos com id maior que o PID dado
- -M PID -> procurar processos com id menor que o PID dado
- -p n -> apenas mostrar n número de processos na tabela
- -r -> ordenar a tabela por ordem inversa
- -w -> ordenar a tabela pela Rate de Escrita
-

O uso de : (dois pontos) após algumas opções é utilizado para garantir que estas tenham de ter argumentos, sendo esta verificação feita pela própria função getopts.

Devido ao uso de argumentos constituídos por várias strings (por exemplo para as opções -s e -e) com o fornecimento de uma data) foi necessária a invocação da função com o seguinte argumento.

```
Argument_checker "$@\""
```

Figura 8 – Chamada à função Argument_checker

Devido a isto, no fim de cada argumento foi colocado uma aspa, que teve de ser removida posteriormente, através do seguinte comando:

```
OPTARG=$(echo $OPTARG | cut -d '"' -f 1) # to be able to have the whole date as a argument , the function
#had to be called with $@\ , which in turn would give the arguments a double quote at the end , so this line
#removes the double quote
```

Figura 9 – Remoção da aspa nos argumentos

Armazenamento das opções e respetivos argumentos

Recorreu-se ao array associativo options declarado anteriormente para guardar a informação sobre as opções e respetivos argumentos.

Caso exista um argumento associado a uma opção (a expressão -z verifica se o comprimento da string OPTARG é 0), o valor no array indexado pela variável $option será o respetivo argumento, tal como se pode ver na figura abaixo.

```
if [[ -z "$OPTARG" ]]; then
    options[$option]="empty"
else
    options[$option]=$OPTARG
fi
```

Caso a opção não tenha um argumento associado, será guardada a string “empty”.

Figura 10 – Gerenciamento dos argumentos

Verificação das opções e argumentos

Sendo assim, foi possível guardar a informação de cada opção e respetivos argumentos numa estrutura de dados, para proceder á sua verificação e uso.

Posteriormente, foram feitas várias verificações das opções e dos argumentos associados.

- Para as opções -c e -u, foi verificado que o argumento fornecido tinha de existir e ser um nome valido, ou seja, o seu valor no array options não podia ser a string “empty” e também não poderia ser um número.

- Para as opções -s e -e, foi garantido que a data fornecida tinha de se encontrar entre aspas, tal como no enunciado do trabalho, e era uma data valida, através do uso do comando date -d. Quando o script for executado com ambas as opções, a data de início (opção -s) tem de ser menor que a data de fim (opção -e).

- Para as opções -m, -M e -p, verificou-se se o argumento associado era um número inteiro positivo.

Também se garantiu que, ao executar o script, teria de ser passado pelo menos um argumento, e que este argumento tinha de ser um número inteiro positivo (tempo para esperar entre as leituras das informações dos processos).

Por fim, confirmou-se também que o último argumento passado não poderia ser um argumento associado a uma opção e que não poderiam ser passadas ao script opções que não sejam suportadas.

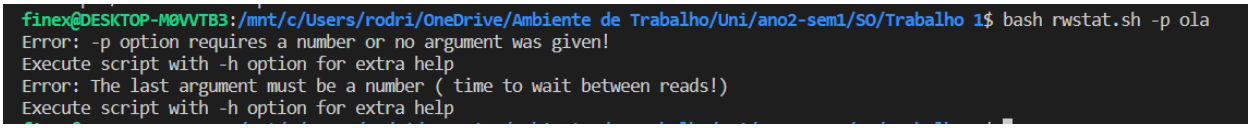

Figura 11 – Mensagens de erro

Caso alguma destas verificações deia resultado negativo, é impresso num terminal uma mensagem de erro e a variável i é atualizada para o valor 1, determinando assim que aconteceu algum erro na verificação das opções e argumentos.

Se após a verificação de todas as opções e argumentos, a variável i tiver o valor 1, o script terminará, imprimindo no terminal as mensagens de erro respetivas, juntamente com o aviso sobre a opção -h, para obter mais ajuda sobre o uso das opções e argumentos no script.

```
Argument_checker "$@\""
if [[ $i == 1 ]] ; then
    exit 1
fi  # if there is an error with the arguments, exit the script
```

Figura 12 – Uso da variável i

Função get_data()

Extração dos valores pretendidos

A função get_data() tem como finalidade a obtenção dos dados sobre os processos, que posteriormente irão ser mostrados no formato de tabela.

Os processos, que contem o seu PID guardado no "arraypids", são iterados um a um, lendo-se os seus ficheiros presentes na pasta /proc.

Antes de cada leitura de um ficheiro, é antes verificado se há permissão para o ler e se o ficheiro existe.

Caso o ficheiro exista e seja possível de ler, é extraido do ficheiro /proc/pid/io os valores de rchar e wchar.

```
for item in "${arraypids[@]}"; do
    if [[ -r /proc/$item/status && -r /proc/$item/io ]]; then
    pid=$item
    Rbytes[$pid]=$( cat /proc/$item/io | awk '{print $2}' | head -n 1  )
    wbytes[$pid]=$( cat /proc/$item/io | awk '{print $2}' | head -n 2 | head -n 1  )
   fi
done
```

Figura 13 – Leitura de rchar e wchar iniciais

Após a primeira leitura, o programa faz um sleep.

Após o sleep, os valores de rchar e wchar voltam a ser lidos novamente e é calculada a diferença entre este valor e o valor inicial. Também são extraidos o nome do processo (comm) , utilizador (user) e a data de início do processo, através dos comandos /proc/pid/comm e ps –o.

Análise dos processos

Ao confirmar-se as condições anteriores, avança-se para a avaliação do processo em questão. Consoante as opções escolhidas na execução do script, os dados do processo poderão ou não ser listados na tabela:

- A expressão -v é utilizada para verificar se o argumento associado a cada opção existe.

- Se os dados dos processos condizerem às opções e argumentos dados pelo utilizador do script, é adicionada uma linha da tabela corretamente formatada com a informação

do processo, num array associativo (chamado processnfo), sendo cada entrada no array indexada pelo PID do processo.

- Caso os dados do processo não correspondam aos argumentos passados pelo utilizador, este processo é saltado, procedendo-se à análise do próximo processo.

- Exemplo: se a opção c foi passada e se o COMM não der match (em regex) à expressão que foi passada como argumento da opção -c, o script salta esse processo e vai para o próximo (dá continue). Se um dos COMM de um processo de match, os dados do mesmo serão adicionados ao array “processinfo” e o for prossegue.

```
if [[ -v options[c] && ! $comm =~ ${options['c']} ]]; then
    continue   # search by name ( regular expression )
fi

# -v = true if variable is set ( has a value)
```

Figura 14 – Verificação de processos

Com isto, conclui-se a análise de um processo, que se repetirá até todos os processos serem analisados.

Implementação de datas e expressões regulares

Para verificar se a data fornecida pelo utilizador era maior que a data de início do processo mínima (opção -s) ou menor que a data de início máxima (opção -e), foi utilizado o comando date -d para converter a data fornecida pelo utilizador para segundos desde 1 de janeiro de 1970 (UNIX time)

Esta data foi então comparada com a data de início do processo, convertida também para UNIX time, para verificar se o seu valor era maior ou menor.

Para se verificar o uso correto com expressões regulares, nas opções -u e -c, foi utilizado o operador =~ da bash, que permite fazer uma verificação de um match através de expressões regulares.

Função prints()

Esta função é a última a ser executada, tendo como objetivo a impressão no terminal da informação recolhida e guardada previamente sobre os processos sobre a forma de uma tabela bem formatada e ordenada conforme as opções -r e -w, passadas ou não na execução do script.

Para começar, a função começa por imprimir o cabeçalho da tabela.

```
function prints() {
    printf "%-27s   %-10s   %-5s    %-10s   %-10s   %-10s   %-10s   %-20s \n" "COMM" "USER" "PID" "RBYTES" "WBYTES" "RATER" "RATEW" "DATE"
```

Figura 15 – Formato da tabela

Foram guardadas as casas que pareceram ser adequadas para cada uma das informações do processo.

Após isto, a função procede à verificação se existe a opção -p com argumento associado.

Caso exista, este argumento é atribuído à variável p.

Caso este não exista, a variável p assume o número de elementos do array associativo “processInfo”.

```
if ! [[ -v options[p] ]]; then
    p=${#processinfo[@]}
#Number of processes to print
else
    p=${options['p']}
fi
```

Figura 16 – Atribuição da variável p

Após isto, é impresssa toda a informação sobre os processos que passaram na verificação, presente no array associativo “processinfo”.

Como em cada posição existente do array está contida a informação para um printf de uma linha da tabela, apenas é necessário imprimir todos os elementos do array.

Estes são posteriormente ordenados:

- Pela RateR crescente, em caso da existência da opção -r
- Pela RateW decrescente, em caso da existência da opção -w
- Pela RateW crescente, em caso de existência de ambas as opções
- Por omissão das duas opções, a tabela é ordenada por RateR decrescente

A informação ordenada na tabela é então passada por pipeline a um “head -n $p”, que apenas imprimirá a informação sobre os primeiros p processos (p sendo a variável falada acima).

```
if [[ -v options[w] && ! -v options[r] ]]; then
    #Rate W decreasingly
    printf '%s \n' "${processinfo[@]}" | sort -rn -k7  | head -n $p

elif [[ -v options[w] && -v options[r] ]]; then
    # RateW increasingly
    printf '%s \n' "${processinfo[@]}" | sort -n -k7  | head -n $p

elif [[ -v options[r] && ! -v options[w] ]]; then
    # RateR increasingly
    printf '%s \n' "${processinfo[@]}" | sort -n -k6 |  head -n $p
else
    # RateR decreasingly
    printf '%s \n' "${processinfo[@]}" | sort -rn -k6 | head -n $p
fi
```

Figura 17 – Print da Tabela

Com isto conclui-se a execução do script.

Fluxo de Execução

Para melhor se perceber como funciona o script, foi elaborado um fluxograma que ilustra todas as etapas da execução do script.

Devido ao tamanho do diagrama, ocuparia uma página inteira e não teria uma visibilidade muito boa.

Sendo assim, uma imagem do diagrama irá ser incluída na pasta entregue, com o nome “FluxoExecução”.

Testes

Para verificar o funcionamento do programa como esperado, foram realizados vários testes, dos quais se pode ver o resultado abaixo.

```
obrist@obrist-IdeaPad-3-15IML05:~/SO/Trabalho1$ ./rwstat.sh -c bash 1
COMM                 USER            PID     RBYTES          WBYTES          RATER           RATEW           DATE
bash                 obrist          7593    0               2346399016      0               2346399016      Dec  1 17:06
```

Figura 18 – Teste da opção -c

```
obrist@obrist-IdeaPad-3-15IML05:~/SO/Trabalho1$ ./rwstat.sh -s "Dec 1 00:00" 1
COMM                 USER            PID     RBYTES          WBYTES          RATER           RATEW           DATE
rwstat.sh            obrist          72859   94423992        98949925        94423992        98949925        Dec  1 17:42
chrome               obrist          7477    1513            204482          1513            204482          Dec  1 17:05
chrome               obrist          7172    709             145099238       709             145099238       Dec  1 17:03
nautilus             obrist          7243    0               15430766        0               15430766        Dec  1 17:03
gnome-terminal-      obrist          7585    0               1435340         0               1435340         Dec  1 17:06
gedit                obrist          67557   0               3286312         0               3286312         Dec  1 17:36
chrome               obrist          7550    0               71033           0               71033           Dec  1 17:05
chrome               obrist          7403    0               250723          0               250723          Dec  1 17:05
chrome               obrist          7351    0               138177129       0               138177129       Dec  1 17:05
chrome               obrist          7185    0               1052191         0               1052191         Dec  1 17:03
```

Figura 19 – Teste da opção -s

```
obrist@obrist-IdeaPad-3-15IML05:~/SO/Trabalho1$ ./rwstat.sh -e "Nov 30 23:59" 1
COMM                 USER            PID     RBYTES          WBYTES          RATER           RATEW           DATE
Discord              obrist          5058    165908          290425660       165908          290425660       Nov 30 14:47
Discord              obrist          5144    62564           282392715       62564           282392715       Nov 30 14:47
chrome               obrist          2148    1733            826085          1733            826085          Nov 30 13:54
chrome               obrist          1856    361             199667838       361             199667838       Nov 30 13:53
Xorg                 obrist          930     146             24196580        146             24196580        Nov 30 13:51
pulseaudio           obrist          904     108             3742206         108             3742206         Nov 30 13:51
gnome-shell          obrist          1207    48              20311799        48              20311799        Nov 30 13:51
xdg-document-po      obrist          1861    24              49605           24              49605           Nov 30 13:53
chrome               obrist          1854    13              1586349         13              1586349         Nov 30 13:53
chrome               obrist          1809    1               2637489623      1               2637489623      Nov 30 13:53
xdg-permission-      obrist          1259    0               20918           0               20918           Nov 30 13:51
```

Figura 20 – Teste da opção -e

```
obrist@obrist-IdeaPad-3-15IML05:~/SO/Trabalho1$ ./rwstat.sh -u obrist 1
COMM                 USER            PID     RBYTES          WBYTES          RATER           RATEW           DATE
rwstat.sh            obrist          83207   92910585        97418771        92910585        97418771        Dec  1 17:44
Discord              obrist          5058    124419          293606688       124419          293606688       Nov 30 14:47
Discord              obrist          5144    62579           287021927       62579           287021927       Nov 30 14:47
pulseaudio           obrist          904     504             3790715         504             3790715         Nov 30 13:51
Xorg                 obrist          930     146             24689967        146             24689967        Nov 30 13:51
gnome-shell          obrist          1207    56              20361663        56              20361663        Nov 30 13:51
gsd-media-keys       obrist          1361    40              4841065         40              4841065         Nov 30 13:51
xdg-document-po      obrist          1861    24              50037           24              50037           Nov 30 13:53
xdg-desktop-por      obrist          1850    24              71290           24              71290           Nov 30 13:53
xdg-permission-      obrist          1259    0               20918           0               20918           Nov 30 13:51
xdg-desktop-por      obrist          1876    0               1956994         0               1956994         Nov 30 13:53
update-notifier      obrist          1615    0               135068615       0               135068615       Nov 30 13:52
tracker-miner-f      obrist          906     0               15704916        0               15704916        Nov 30 13:51
systemd              obrist          886     0               160121448       0               160121448       Nov 30 13:51
nautilus             obrist          7243    0               16165074        0               16165074        Dec  1 17:03
nacl_helper          obrist          1828    0               10208           0               10208           Nov 30 13:53
```

Figura 21 – Teste da opção -u

```
obrist@obrist-IdeaPad-3-15IML05:~/SO/Trabalho1$ ./rwstat.sh -m 1800 -M 1820 1
COMM                            USER            PID     RBYTES          WBYTES          RATER           RATEW           DATE
chrome                          obrist          1809    13              2639090586      13              2639090586      Nov 30 13:53
chrome_crashpad                 obrist          1820    0               7700            0               7700            Nov 30 13:53
chrome_crashpad                 obrist          1818    0               7700            0               7700            Nov 30 13:53
cat                             obrist          1816    0               5412            0               5412            Nov 30 13:53
cat                             obrist          1815    0               4292            0               4292            Nov 30 13:53
```

Figura 22 – Teste das opções -m e -M

```
obrist@obrist-IdeaPad-3-15IML05:~/SO/Trabalho1$ ./rwstat.sh -r -w -p 10 1
COMM                            USER            PID     RBYTES          WBYTES          RATER           RATEW           DATE
cat                             obrist          1815    0               4292            0               4292            Nov 30 13:53
cat                             obrist          1816    0               5531            0               5531            Nov 30 13:53
chrome_crashpad                 obrist          1818    0               7700            0               7700            Nov 30 13:53
chrome_crashpad                 obrist          1820    0               7700            0               7700            Nov 30 13:53
nacl_helper                     obrist          1828    0               10208           0               10208           Nov 30 13:53
chrome                          obrist          105200  1518            16721           1518            16721           Dec  1 19:04
xdg-permission-                 obrist          1259    0               21014           0               21014           Nov 30 13:51
gnome-session-c                 obrist          1186    0               24478           0               24478           Nov 30 13:51
gsd-disk-utilit                 obrist          1420    0               24977           0               24977           Nov 30 13:51
gvfs-goa-volume                 obrist          983     0               26886           0               26886           Nov 30 13:51
```

Figura 23 – Teste das opções de ordenação (-p , -r , -w)

Erros

```
obrist@obrist-IdeaPad-3-15IML05:~/SO/Trabalho1$ ./rwstat.sh -p 3
Error: The last argument is the argument to the option -p , not the sleep time!!
Example of correct usage: ./rwstat.sh -p "-p argument" 7
```

Figura 24 – Tentativa de execução sem passagem de intervalo de tempo entre leituras

```
obrist@obrist-IdeaPad-3-15IML05:~/SO/Trabalho1$ ./rwstat.sh
Atleast one argument must be passed ( time to wait between reads!)
Execute script with -h option for extra help
```

Figura 25 – Execução do script sem argumentos

```
obrist@obrist-IdeaPad-3-15IML05:~/SO/Trabalho1$ ./rwstat.sh -p ola
Error: -p option requires a number or no argument was given!
Execute script with -h option for extra help
Error: The last argument must be a number ( time to wait between reads!)
Execute script with -h option for extra help
```

Figura 26 – Execução do script com argumento errado e sem intervalo de tempo

```
obrist@obrist-IdeaPad-3-15IML05:~/SO/Trabalho1$ ./rwstat.sh -s "Nov 30 10:00" -e "Nov 28 23:04" 1
Error: The start date must be before the end date
```

Figura 27 – Data de início maior que a data de fim

```
obrist@obrist-IdeaPad-3-15IML05:~/SO/Trabalho1$ ./rwstat.sh -p
Error: -p option requires a number or no argument was given!
Execute script with -h option for extra help
Error: The last argument must be a number ( time to wait between reads!)
Execute script with -h option for extra help
```

Figura 28 – Uso de uma opção sem argumento

```
obrist@obrist-IdeaPad-3-15IML05:~/SO/Trabalho1$ ./rwstat.sh -s "JJJ 21:32" 1
Error: Invalid date (double quotes are needed) or no date was given at all!
Execute script with -h option for extra help
```

Figura 29 – Uso de uma data inválida

Conclusão

Com base nos testes feitos, pode-se concluir que a resolução deste trabalho foi feita com sucesso, sendo todas as funcionalidades implementadas com o resultado esperado.

Durante o trabalho, ocorreram vários problemas, que acabaram por ser resolvidos utilizando exemplos de aulas práticas e pesquisa na internet.

Também foi essencial o comando “man” da bash, para entender qual a funcionalidade e utilidade de alguns comandos utilizados neste trabalho.

Este trabalho enriqueceu o nosso conhecimento e curiosidade em bash, e também ajudou a perceber melhor como funcionam processos e o sistema de ficheiros do sistema operativo Linux.

Bibliografia

- https://stackoverflow.com
- https://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_01.html
- https://unix.stackexchange.com
- https://www.unix.com
- Aulas teóricas/práticas