Sistemas Operativos

Monitorização de interfaces em bash

Trabalho nº1

2021/2022

Vânia Inês Magalhães Morais - 10238
Yanis Marina Faquir - 100181

Índice

Introdução

O seguinte relatório tem como objetivo descrever qual a abordagem usada para a resolução do problema proposto neste trabalho.

O objetivo deste é o desenvolvimento de um script, em bash, que apresenta estatísticas sobre a quantidade de dados transmitidos e recebidos nas interfaces de rede selecionadas e sobre as respetivas taxas de transferência.

O script **netifastat.sh** permite ver a quantidade de dados transmitidos e recebidos nas interfaces de rede selecionadas e as respetivas taxas de transferência para períodos de tempo pré-estabelecidos.

Estrutura do código- Definição de variáveis

A partir deste ponto, iremos mostrar partes do código e explicar o raciocínio que nos levou ao código em questão.

```
flagP=0;
flagC=0;
flagDados=0;
flagL=0;
byte=0;


flagSort=0;          # verificaçao
flagReverse=0;       # verificar default

sort="sort "
order=" -n "         # ordem por default
```

Fig. 1- Definição de Variáveis

Para começar definimos algumas variáveis que nos vão ser úteis mais à frente. Depois declaramos uma mensagem de texto que aparecerá caso o utilizador coloque algum parâmetro de forma errada ou não coloque o parâmetro obrigatório (número de segundos).

Estrutura do código- Parâmetros de entrada

De seguida tratamos dos parâmetros de entrada. Para tal, usamos a estrutura **getopts**. Esta estrutura usa opções curtas, que são um traço (“-”) e uma letra ou digito (por exemplo, -r ou -t) e altera o valor de 0 para 1 de uma “flag”, para mostrar que essa opção foi usada. Também se pode fazer combinações de algumas opções, como iremos explicar mais à frente.

Filtros de procura

- **-c**: Aqui há seleção das interfaces a partir de uma expressão regular, onde a coluna filtrada é a **NETIF** (onde se encontra o nome de cada interface). Também é obrigatória a presença de uma expressão após a seleção do parâmetro.

```
c) #selecção dos interfaces a visualizar pode ser realizada através de uma expressão regular
    c="${OPTARG}"

    if [[ $flagC == 1 ]]; then
        echo "ERRO: -c <OPTION> só pode ser selecionado uma vez!"
        exit 1
    fi

    flagC=1;
;;
```

Fig. 2- Filtro de Procura nº1

- **-p**: Neste parâmetro o utilizador escolhe o número de interfaces que quer visualizar. É exigido que tenha um número, que este seja válido e só pode ser selecionado uma vez por cada vez que se corre o script.

```
p) # número de interfaces a visualizar
    p="${OPTARG}"

    if [[ ! "${p}" =~ ^[0-9] ]]; then
        printf "\n  ERRO: Número de interfaces inválido!"
        message
        exit 1
    fi

    if [[ $flagP == 1 ]]; then
        printf "\n  ERRO: -p <OPTION> só pode ser selecionado uma vez!"
        message
        exit 1
    fi

    flagP=1;
;;
```

Fig. 3- Filtro de Procura nº2

Filtros de Visualização

- **-b**: Esta opção faz com que os valores estejam em bytes.

```
b) #mostrar os dados em bytes

    if [[ $flagDados == 0 ]]; then
        flagDados=1;
        byte=1;
    else
        echo "ERRO: Não se pode selecionar mais do que um tipo de visualisação!"
        message
        exit 1
    fi
;;
```

Fig. 4- Filtro de Visualização nº1

- **-k**: Esta opção faz com que os valores estejam em kilobytes.

```
k) #mostrar os dados em kilobytes

    if [[ $flagDados == 0 ]]; then
        flagDados=1;
        byte=1024;
    else
        echo "ERRO: Não se pode selecionar mais do que um tipo de visualisação!"
        message
        exit 1
    fi
;;
```

Fig. 5-Filtro de Visualização nº2

- **-m**: Esta opção faz com que os valores estejam em megabytes.

```
m) #mostrar os dados em megabytes

    if [[ $flagDados == 0 ]]; then
        flagDados=1;
        byte=1000000;
    else
        echo "ERRO: Não se pode selecionar mais do que um tipo de visualisação!"
        message
        exit 1
    fi
;;
```

Fig. 6- Filtro de Visualização nº3

Em todos os trechos de código deste filtro, existe uma variável chamada “bytes”, que mostra quantos bytes são cada uma das opções. Esta variável vai-nos ser útil mais tarde para a conversão dos valores.

Aqui também não pode ser usado mais do que um tipo de visualização, daí serem usados os **if’s**.

Ordenação das colunas

```
v) #reverse

    if [[ $flagReverse == 0 ]]; then
        order=" -rn "
        flagReverse=1
    else
        printf "\n  ERRO: Só pode utilizar '-v' no máximo uma vez!"
        message
        exit 1
    fi
;;
```

Fig. 7- Ordenação das Colunas nº1

- **-v**: Com este parâmetro, o utilizador reverte a ordem *default*. Esta opção é a única pode ser usada com outra do mesmo tipo.
- **-t**: Usado para que os valores de **TX** sejam ordenados de forma decrescente.
- **-r**: Usado para que os valores de **RX** sejam ordenados de forma decrescente.
- **-T**: Usado para que os valores de **TTRATE** sejam ordenados de forma decrescente.
- **-R**: Usado para que os valores de **RRATE** sejam ordenados de forma decrescente.

Todas as opções acima têm a variável “flagSort” com o mesmo nome, para que não seja usada mais do que uma opção desta família com o -v.

```
t)  # sort on TX

    if [[ $flagSort == 0 ]]; then
        flagSort=1;
        sort+=" -k2 ";
        order=" -n "
    else
        printf "\n  ERRO: Não se pode selecionar mais que 1 tipo de ordenação!"
        message
        exit 1
    fi
;;

r)  # sort on RX

    if [[ $flagSort == 0 ]]; then
        flagSort=1;
        sort+=" -k3 ";
        order=" -n "
    else
        printf "\n  ERRO: Não se pode selecionar mais que 1 tipo de ordenação!"
        message
        exit 1
    fi
;;
```

Fig. 8- Ordenação das colunas nº2 e 3

```
T)  # sort on TRATE

    if [[ $flagSort == 0 ]]; then
        flagSort=1;
        sort+=" -k4 ";
        order=" -n "
    else
        printf "\n  ERRO: Não se pode selecionar mais que 1 tipo de ordenação!"
        message
        exit 1
    fi
;;

R)  # sort on RRATE

    if [[ $flagSort == 0 ]]; then
        flagSort=1;
        sort+=" -k5 ";
        order=" -n "
    else
        printf "\n  ERRO: Não se pode selecionar mais que 1 tipo de ordenação!"
        message
        exit 1
    fi
;;
```

Fig. 9- Ordenação de colunas nº4 e 5

Opção extra

- **-l**: Esta opção faz com que o script funcione em loop, sem fim, imprimindo a cada **s** (parâmetro obrigatório a ser posto no fim) segundos nova informação. Com esta opção também são acrescentadas duas colunas à tabela imprimida: **TXTOT** e **RXTOT**. Estas novas colunas indicam as quantidades de dados transmitidos e recebidos desde o início da execução do script, enquanto que as já existentes são sempre relativas aos últimos **s** segundos.

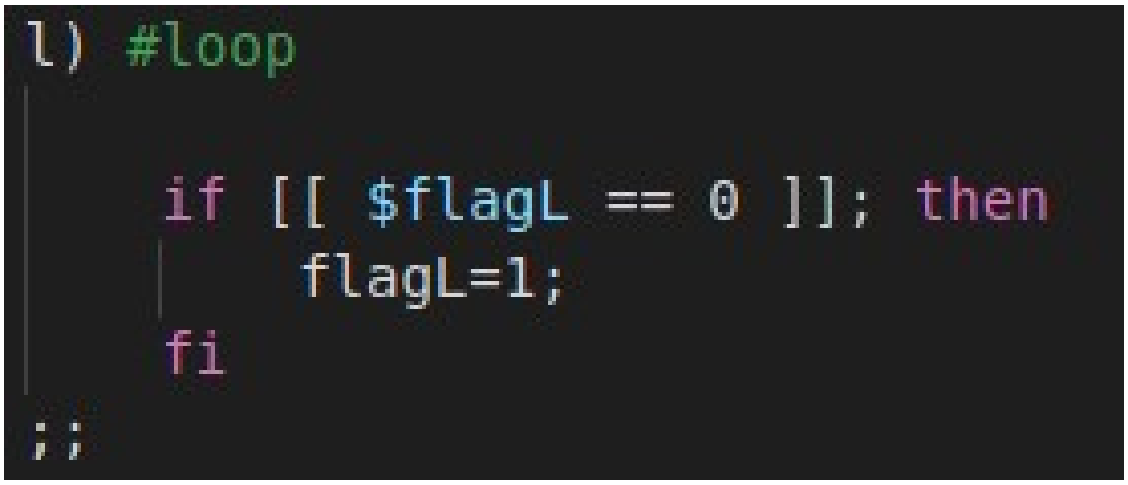

Fig. 10- Opção extra

Aqui, a variável “flagL” vai ser útil para a criação da função que dará origem ao loop.

Por fim, a última opção do *getops* fica ativa quando não é respeitada alguma das regras a ser seguidas nos parâmetros acima, ou quando o utilizador não usa parâmetros obrigatório (segundos).

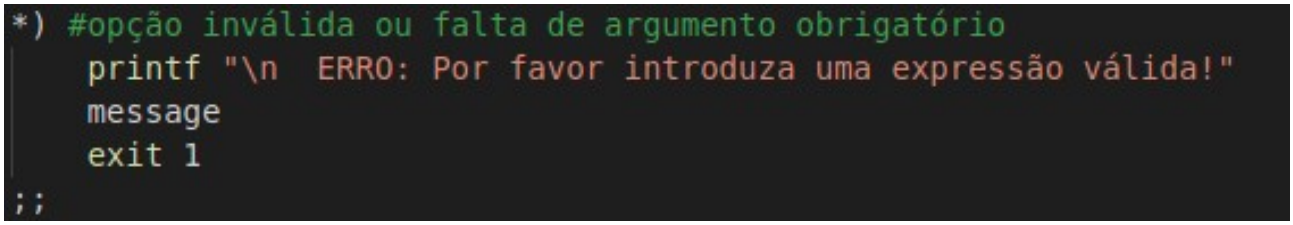

Fig. 11- Opção final

Estrutura do código- Validações e Verificações

De modo a que não haja erros ao longo do script, devido parâmetros postos pelo utilizador, é necessário fazer algumas verificações/validações de dados.

```
#validações

if [[ $flagSort == 1 && $flagReverse == 0 ]]; then
    order="-r "
fi

if [[ $flagP == 1 && $flagReverse == 1 ]]; then
    order="-r"
fi

re='^[0-9]+([.][0-9]+)?$'
if [[ ${@: -1} =~ $re ]]; then      # verificar se o último argumento é um número
    segundos=${@: -1}
else
    printf "\n  ERRO: Por favor introduza uma expressão válida!"
    message
    exit 1
fi

if [[ ${#@} -gt 1 ]]; then      # verificar que o único argumento excluindo as opções é o $segundos
    printf "\n  ERRO: Por favor introduza uma expressão válida!"
    message
    exit 1
fi
```

Fig. 12- Validações e Verificações

O 1º **if** verifica se os parâmetros escolhidos são os segundos e algum dos **sorts**. Se sim, a tabela é imprimida em função do **sort**.

Já o 2º **if** verifica se os parâmetros escolhidos são os segundos, o -p e o -v. Se sim, a tabela é imprimida com a ordem das redes contrária à ordem alfabética.

Depois temos as verificações para confirmar se o último argumento é o **$segundos** e se este é um número inteiro positivo.

Estrutura do código- Função get_dados()

Na função get_dados() é feito tudo aquilo que é necessário ao funcionamento do script, isto é, lá está contido a extração dos dados necessários e a implementação de todos os parâmetros.

```
rede=($(ifconfig | grep -w mtu | awk '{print $1}')) #linha com nome da rede
dadosRX=($(ifconfig | grep -w "RX packets" | awk '{print $3}')) #guarda volor do bytes RX
dadosTX=($(ifconfig | grep -w "TX packets" | awk '{print $3}'))  #guarda volor do bytes tx
```

Fig. 13-Recolha de dados

A partir do **ifconfig**, retiramos os nomes das redes, os dados de RX e de TX. Usamos **grep** e **awk** para nos facilitar na recolha, uma vez que são comandos que filtram as linhas e as colunas, respetivamente.

```
for (( k=0; k<${#rede[@]}; k++ )); #percorrE
do
    if [[ $flagDados == 1 ]]; then
        dadosTX[k]=$(echo "scale=4; ${dadosTX[k]}/$byte" | bc -l)
        dadosRX[k]=$(echo "scale=4; ${dadosRX[k]}/$byte" | bc -l)
        dados2TX[k]=$(echo "scale=4; ${dados2TX[k]}/$byte" | bc -l)
        dados2RX[k]=$(echo "scale=4; ${dados2RX[k]}/$byte" | bc -l)
    fi

    difTX[k]=$(echo "scale=4; ${dados2TX[k]} - ${dadosTX[k]}" | bc -l)
    TRATE[k]=$(echo "scale=4; ${difTX[k]} / $segundos" | bc -l)

    difRX[k]=$(echo "scale=4; ${dados2RX[k]} - ${dadosRX[$k]}" | bc -l)
    RRATE[k]=$(echo "scale=4; ${difRX[k]}  / $segundos" | bc -l)
    i=$(($i+1))
done
```

Fig. 14- Obtenção dos restantes dados

Neste ciclo **for**, temos um **if** que vai converter os valores, caso uma das opções de visualização seja escolhida. Após a conversão, é calculado o valor do TX e do RX, resultantes da diferença entre os dados recolhidos depois do **sleep** e antes do mesmo. É também calculado o **TRATE** e o **RRATE**, dividindo os valores anteriores pela variável **$segundos**.

```
if [[ $flagP == 1 ]]; then
    if [[  $i -lt $p ]]; then
        printf "\n  ERRO: Impossível mostrar %d processos com as opções selecionadas!" "$p"
        message; exit 1
    elif [[ $flagC == 1 ]]; then    # erro se houver mais processos pedidos em -p q retornados de -c
        if  [[ $cc -lt $p ]]; then
            printf "\n  ERRO: Impossível mostrar %d processos com as opções selecionadas!" "$p"
        message; exit 1
        fi
    fi
fi
```

Fig. 15- Mais algumas validações

Aqui é verificado se a quantidade de interfaces escolhidas pelo utilizador é compatível com o número de resultados da procura no parâmetro -c. Se não, é imprimida uma mensagem de erro e o programa termina.

```
    printf "%10s %10s %10s %10s %10s %10s %10s \n" "NETIF" "TX" "RX" "TRATE" "RRATE" "TXTOT" "RXTOT"
    echo ""> dados.txt
    for ((k=0; k<${#rede[@]}; k++)); #percorrE
    do
        TXTOT[k]=$(echo "scale=4; ${TXTOT[k]} + ${difTX[k]}" | bc -l)
        RXTOT[k]=$(echo "scale=4; ${RXTOT[k]} + ${difRX[k]}" | bc -l)

        printf "%10s %10s %10s %10s %10s %10s %10s \n" ${rede[$k]} ${difTX[$k]} ${difRX[$k]} ${TRATE[$k]} ${RRATE[$k]} ${TXTOT[$k]} ${RXTOT[$k]} >> dados.txt
    done
```

Fig. 16- printf da tabela

Neste trecho de código é descoberto o valor de **TXTOT** e **RXTOT**, é imprimida a tabela e todos os seus dados são guardados num ficheiro chamado “dados.txt”. Este ficheiro vai nos ser útil para mais à frente imprimir os valores de acordo com cada parâmetro a partir do comando **cat**, como podemos observar neste exemplo:

```
elif [[ $flagSort == 1 ]]; then
    if [[ $flagReverse == 1 ]]; then
        cat dados.txt | grep -w $c | ${sort} -n
    else
        cat dados.txt | grep -w $c | ${sort} ${order}
    fi
```

Fig. 17-Exemplo

Estrutura do código- loop

Por último, de maneira a chamar a função **get_dados()**, foi feito um **if** para se verificar se o parâmetro -l tinha sido utilizado. Se sim, é feito um loop infinito através de um ciclo **while**. Se não, a função é somente chamada:

```
if [[ $flagL == 1 ]]; then
    while : ; do
        get_dados

    done
else
    get_dados
fi
```

Fig. 17- Loop

Testes realizados

Agora faremos alguns testes e mostraremos as respostas quando os parâmetros são válidos e quando não o são.

Respostas a comandos válidos

```
vania@vania-VivoBook-ASUSLaptop-X571LI-F571LI:~/Desktop/SO/trabalho1$ ./netifstat.sh -t -l 10
    NETIF         TX         RX      TRATE      RRATE      TXTOT      RXTOT
    wlo1:         93        166     9.3000    16.6000         93        166
      lo:          8          8      .8000      .8000          8          8
  virbr0:          0          0          0          0          0          0
  enp3s0:          0          0          0          0          0          0


    NETIF         TX         RX      TRATE      RRATE      TXTOT      RXTOT
    wlo1:        810       1676    81.0000   167.6000        903       1842
      lo:          0          0          0          0          8          8
  virbr0:          0          0          0          0          0          0
  enp3s0:          0          0          0          0          0          0
```

Fig. 18- sort na coluna TX dentro do loop

```
vania@vania-VivoBook-ASUSLaptop-X571LI-F571LI:~/Desktop/SO/trabalho1$ ./netifstat.sh -k -R -l 10
    NETIF         TX         RX      TRATE      RRATE      TXTOT      RXTOT
    wlo1:      .0068      .0068      .0006      .0006      .0068      .0068
  virbr0:          0          0          0          0          0          0
      lo:          0          0          0          0          0          0
  enp3s0:          0          0          0          0          0          0


    NETIF         TX         RX      TRATE      RRATE      TXTOT      RXTOT
    wlo1:      .0156      .0146      .0015      .0014      .0224      .0214
  virbr0:          0          0          0          0          0          0
      lo:          0          0          0          0          0          0
  enp3s0:          0          0          0          0          0          0


    NETIF         TX         RX      TRATE      RRATE      TXTOT      RXTOT
    wlo1:      .0156      .0176      .0015      .0017      .0380      .0390
      lo:      .0039      .0039      .0003      .0003      .0039      .0039
  virbr0:          0          0          0          0          0          0
  enp3s0:          0          0          0          0          0          0
```

Fig. 19- dados em kiloBytes, com sort na coluna RRATE dentro do loop

```
vania@vania-VivoBook-ASUSLaptop-X571LI-F571LI:~/Desktop/SO/trabalho1$ ./netifstat.sh -c "l.*" 10
    NETIF         TX         RX      TRATE      RRATE
      lo:          4          4      .4000      .4000
```

Fig. 20- seleção de uma interface através de uma expressão regular

Respostas a comandos inválidos

```
vania@vania-VivoBook-ASUSLaptop-X571LI-F571LI:~/Desktop/SO/trabalho1$ ./netifstat.sh -t -T 10

        ERRO: Não se pode selecionar mais que 1 tipo de ordenação!
        ---------------------------------------------- OPÇÕES VÁLIDAS ----------------------------------------------

        ./netifstat.sh <OPTION> : onde OPTION é número de segundos que serão usados para calcular as taxas de transferência

        Nota: O último argumento passado terá de ser o número de segundos obrigatoriamente!

            Filtros de procura:
                -p <OPTION> : número de interfaces a visualizar (OPTION=number)
                -c <OPTION> : selecionar interfaces através de uma expressão regular (OPTION=REGEX EXPRESSION)

        Filtros de visualisação:
                -b    : todos os dados estarão em bytes
                    -k    : todos os dados estarão em kilobytes
                -m    : todos os dados estarão em megabytes

        Ordenação das colunas:
                -v          : reverse order
                -r          : sorts on RX
                -t          : sorts on TX
                -T          : sorts on TRATE
                -R          : sorts on RRATE

        Opção extra:
                -l      : loop onde a cada s segundos é imprimida nova informação
```

Fig. 21- foram colocados 2 parâmetros do tipo sort

```
vania@vania-VivoBook-ASUSLaptop-X571LI-F571LI:~/Desktop/SO/trabalho1$ ./netifstat.sh -p 3 -c 0 10

        ERRO: Impossível mostrar 3 processos com as opções selecionadas!
        ------------------------------------------------ OPÇÕES VÁLIDAS ------------------------------------------------

        ./netifstat.sh <OPTION> : onde OPTION é número de segundos que serão usados para calcular as taxas de transferência

        Nota: O último argumento passado terá de ser o número de segundos obrigatoriamente!

            Filtros de procura:
                -p <OPTION> : número de interfaces a visualizar (OPTION=number)
                -c <OPTION> : selecionar interfaces através de uma expressão regular (OPTION=REGEX EXPRESSION)

        Filtros de visualisação:
                -b    : todos os dados estarão em bytes
                    -k    : todos os dados estarão em kilobytes
                -m    : todos os dados estarão em megabytes

        Ordenação das colunas:
                -v          : reverse order
                -r          : sorts on RX
                -t          : sorts on TX
                -T          : sorts on TRATE
                -R          : sorts on RRATE

        Opção extra:
                -l      : loop onde a cada s segundos é imprimida nova informação
```

Fig. 22- foram inseridas mais interfaces do que as existentes devido à expressão no -c

```
vania@vania-VivoBook-ASUSLaptop-X571LI-F571LI:~/Desktop/SO/trabalho1$ ./netifstat.sh -f 10

        ERRO: Por favor introduza uma expressão válida!
        ------------------------------------------------ OPÇÕES VÁLIDAS ------------------------------------------------

        ./netifstat.sh <OPTION> : onde OPTION é número de segundos que serão usados para calcular as taxas de transferência

        Nota: O último argumento passado terá de ser o número de segundos obrigatoriamente!

            Filtros de procura:
                -p <OPTION> : número de interfaces a visualizar (OPTION=number)
                -c <OPTION> : selecionar interfaces através de uma expressão regular (OPTION=REGEX EXPRESSION)

        Filtros de visualisação:
                -b    : todos os dados estarão em bytes
                    -k    : todos os dados estarão em kilobytes
                -m    : todos os dados estarão em megabytes

        Ordenação das colunas:
                -v          : reverse order
                -r          : sorts on RX
                -t          : sorts on TX
                -T          : sorts on TRATE
                -R          : sorts on RRATE

        Opção extra:
                -l      : loop onde a cada s segundos é imprimida nova informação
```

Fig. 23- Foi inserido um parâmetro não existente

Conclusão

Ao longo deste trabalho conseguimos entender melhor como funciona a **bash**. Consolidamos conhecimentos tanto a nível teórico como a nível prático.

Percebemos também que ao escrever o relatório fizemos mais esforço para entender todos os comandos, de modo a poder explicar da forma mais clara o que estava a acontecer no script.

Foi difícil, mas este trabalho ensinou-nos a ser persistentes e a trabalhar em grupo da melhor forma possível.

Referências

- https://www.computerhope.com/unix/bash/getopts.htm – Usado para entender-mos melhor o que é o getopts
- https://www.linuxforce.com.br/comandos-linux/comandos-linux-comando-ifconfig/ - Usado para entender todos os detalhes que existem num ifconfig