

05/12/2021



Sistemas Operativos – Trabalho 1

MONITORIZAÇÃO DE INTERFACES DE REDE EM BASH

Professor – Nuno Lau (nunolau@ua.pt)

Manuel Diaz 103645 P6 (manu.guerra.diaz@ua.pt)

Tiago Carvalho 104142 P6 (tiagogcarvalho@ua.pt)

ÍNDICE

1. Introdução ao Trabalho -----	3
2. Estrutura do Código Desenvolvido -----	4
2.1. Declaração de variáveis globais -----	4
2.2. Tratamento das opções passadas -----	5
2.2.1. Função usage -----	5
2.2.2. Validação do Último Argumento -----	6
2.2.3. While getopt -----	6
2.2.4. Variável \$ctr e Controlo do Número de Argumentos -----	9
2.2.5. Função getTable Dependendo da Opção -l -----	9
3. Processos de Execução -----	11
3.1. Função getTable -----	11
3.1.1. Condições dos Ciclos For -----	11
3.1.2. Primeiro Ciclo For (Primeiro Pedido da Quantidade de Dados) -----	11
3.1.3. Segundo Ciclo For (Cálculo dos Valores Necessários à Impressão) -----	12
4. Impressão dos dados -----	14
4.1. Impressão do Cabeçalho da Tabela -----	14
4.2. Impressão dos Dados das Interfaces de Rede -----	14
5. Resultados -----	15
5.1. Resultados Válidos -----	15
5.2. Resultados Inválidos -----	17
6. Conclusão -----	21
7. Bibliografia -----	22

1 – Introdução ao Trabalho

No âmbito da disciplina de Sistemas Operativos foi-nos proposta a realização de um trabalho prático, que nos permita obter informação sobre a quantidade de dados transmitidos e recebidos nas interfaces de rede selecionadas, e sobre as respetivas taxas de transferência. Para isso, iremos programar o *script* netifstat.sh, de modo que este nos permita visualizar os diferentes processos que estão a correr na nossa máquina.

A linguagem de programação utilizada no trabalho foi bash e, no nosso caso, a implementação do código foi feita no VS Code. A ferramenta “live share” disponível no mesmo permitiu realizar a partilha de código entre os membros do grupo e a elaboração do código no mesmo *script*. Para além disso, a maioria das alterações ao código foram registadas no GitHub (tiagosora/SO).

De acordo com o guião, o *script* netifstat.sh terá de ser executado passando-lhe um, ou vários, argumentos e o programa terá de apresentar as informações que se pretendem de acordo com os mesmos. Para isso, a nossa ideia inicial passou por criar arrays associativos, uma vez que nos permitem identificar cada elemento por uma *key* e um *value*. Assim, e com a utilização dos comandos presentes no guião, que nos permitem fazer a filtragem da informação, conseguiremos guardar e formatar toda a informação dos processos. Depois, iremos criar expressões condicionais para o programa tratar a informação e os valores de acordo com as opções passadas no terminal.

Com a realização deste trabalho esperamos alargar os nossos conhecimentos acerca da programação em bash e perceber mais sobre o funcionamento do sistema operativo Linux – Ubuntu.

2 – Estrutura do Código Desenvolvido

2.1 – Declaração De Variáveis Globais

Inicialmente, como variáveis globais começámos por criar os arrays associativos, onde cada elemento é representado por uma *key* e um *value* e, desta forma, para nos referirmos a um determinado “valor” utilizamos uma “chave”.

```
# Inicialização de Arrays
declare -A optsOrd # Array Associativo para tratamento das opções seleccionadas. Contem informações sobre os argumentos passados.
declare -A rx # Array Associativo para guardar os valores de RX de cada interface de rede, na unidade de visualização desejada.
declare -A rxb1 # Array Associativo para guardar os valores de RX1 de cada interface de rede, em bytes.
declare -A rxb2 # Array Associativo para guardar os valores de RX2 de cada interface de rede, em bytes.
declare -A tx # Array Associativo para guardar os valores de TX de cada interface de rede, na unidade de visualização desejada.
declare -A txb1 # Array Associativo para guardar os valores de TX1 de cada interface de rede, em bytes.
declare -A txb2 # Array Associativo para guardar os valores de TX2 de cada interface de rede, em bytes.
declare -A trate # Array Associativo para guardar os valores de TRATE de cada interface de rede, na unidade de visualização desejada.
declare -A rrate # Array Associativo para guardar os valores de RRATE de cada interface de rede, na unidade de visualização desejada.
declare -A txtot # Array Associativo para guardar os valores de TXTOT de cada interface de rede, na unidade de visualização desejada.
declare -A rxtot # Array Associativo para guardar os valores de RXTOT de cada interface de rede, na unidade de visualização desejada.
```

Img 1 – Inicialização de arrays

optsOrd	Array associativo para tratamento das opções seleccionadas. Contém informação sobre os argumentos passados.
rx	Array associativo para guardar os valores de RX de cada interface de rede, na unidade de visualização desejada.
rxb	Array associativo para guardar os valores de RX de cada interface de rede, em bytes.
rxb1	Array associativo para guardar os valores de RX1 de cada interface de rede, em bytes.
rxb2	Array associativo para guardar os valores de RX2 de cada interface de rede, em bytes.
tx	Array associativo para guardar os valores de TX de cada interface de rede, na unidade de visualização desejada.
txb	Array associativo para guardar os valores de TX de cada interface de rede, em bytes.
txb1	Array associativo para guardar os valores de TX1 de cada interface de rede, em bytes.
txb2	Array associativo para guardar os valores de TX2 de cada interface de rede, em bytes.
trate	Array associativo para guardar os valores de TRATE de cada interface de rede, na unidade de visualização desejada.
rrate	Array associativo para guardar os valores de RRATE de cada interface de rede, na unidade de visualização desejada.
txtot	Array associativo para guardar os valores de TXTOT de cada interface de rede, na unidade de visualização desejada.
rxtot	Array associativo para guardar os valores de RXTOT de cada interface de rede, na unidade de visualização desejada.

Para além de arrays, também utilizámos algumas variáveis globais que servem maioritariamente para validação de argumentos.

```
# Inicilização de variáveis
nre="^[0-9]+(\.[0-9]*)?$" # Expressão regular usada para números.
i=0 # Usada para verificar a condição de uso de apenas um dos -b, -k ou -m.
d=0 # Usada para verificar a condição do argumento passado entre -b, -k ou -m.
m=0 # Usada para verificar a condição de uso de apenas um dos -t, -r, -T ou -R.
l=0 # Usada para transportar valor do loop de -l.
p=-1 # Usada para transportar o número de interfaces a visualizar de -p.
ctr=1 # Usada para calcular o valor de controlo dos argumentos.
k=1 # Usada para determinar a coluna da tabela relevante à ordenação.
c="" # Usada para transportar a expressão regular passada em -c
reverse="" # Usada para alternar a ordenação entre decrescente e crescente.
t=${@: -1} # Usada para guardar o último argumento e usá-lo em todo o programa.
```

Img 2 – Inicialização de variáveis

nre	Expressão regular usada para números.
i	Usada para verificar a condição de uso de apenas um dos -b, -k ou -m.
d	Usada para verificar a condição do argumento passado entre -b, -k ou -m.
m	Usada para verificar a condição de uso de apenas um dos -t, -r, -T ou -R.
l	Usada para transportar o valor do loop de -l.
p	Usada para transportar o número de interfaces a visualizar de -p.
ctr	Usada para calcular o valor de controlo dos argumentos.
k	Usada para determinar a coluna da tabela relevante à ordenação.
c	Usada para transportar a expressão regular passada em -c.
reverse	Usada para alterar a ordenação entre decrescente e crescente.
t	Usada para guardar o último argumento e usá-lo em todo o programa.

2.2 – Tratamento das opções passadas

2.2.1 – Função usage

Esta função serve de apoio ao utilizador, caso este faça uma má utilização do programa. Quase o comando de execução do programa seja invalido, esta função será executada, imprimindo um menu que mostra como usar o programa.

```
function usage() { # Menu de execução do programa.
    echo "Menu de Uso e Execução do Programa."
    echo "  -c [NETIF] : Seleção das interfaces de rede, [NETIF], a visualizar através de uma expressão regular."
    echo "  -b          : Visualização das quantidades em bytes."
    echo "  -k          : Visualização das quantidades em kilobytes."
    echo "  -m          : Visualização das quantidades em megabytes."
    echo "  -p [n]      : Número, [n], de interfaces de redes a visualizar."
    echo "  -t          : Ordenação da tabela por TX (decrescente)."
    echo "  -r          : Ordenação da tabela por RX (decrescente)."
    echo "  -T          : Ordenação da tabela por TRATE (decrescente)."
    echo "  -R          : Ordenação da tabela por RRATE (decrescente)."
    echo "  -v          : Ordenação reversa (crescente)."
    echo "  -l          : Loop de execução do programa a cada [s] segundos."
    echo "ALERTAS -> As opções -t,-r,-T,-R não podem ser utilizadas em simultâneo."
    echo "          O último argumento passado tem de o período de tempo desejado (segundos)."
}
```

Img 3 – Função usage

A opção **-c**, permitirá ao utilizador a seleção das interfaces de rede que pretende visualizar através de uma expressão regular. As opções **-b**, **-k** e **-m**, possibilita o utilizador escolher a unidade de visualização desejada (bytes, kilobytes ou megabytes, respetivamente). As opções **-t**, **-r**, **-T** e **-R**, que servem apenas para ordenar os processos de acordo com as preferências do utilizador (TX, RX, TRATE ou RRATE), por ordem decrescente. A opção **-v**, concede a possibilidade de ordenação reversa, ou seja, por ordem crescente. A opção **-l**, que permite a execução do programa em loop a cada [s] segundos introduzidos pelo utilizador.

2.2.2 – Validação do último argumento

Antes de qualquer tratamento de opções verifica-se o último argumento passado pelo utilizador no comando da execução. Começámos por verificar a existência ou não do último argumento (parâmetro obrigatório) através da condição *if*.

```
# Verificação da existência do último argumento.
if [[ $# == 0 ]]; then
    echo "Necessário, pelo menos, o período de tempo desejado (segundos). Ex -> ./netifstat.sh 10"
    usage # Menu de execução do programa.
    exit 1 # Terminar o programa
fi
```

Img 4 – Verificação da existência do último argumento

Para além da sua existência, também é obrigatório que este seja um número. Então para isso, criámos outra condição para confirmar se este faz match com a variável global *\$nre* (expressão regular para números).

```
# Verificação do último argumento.
if ! [[ $t =~ $nre ]]; then
    echo "O último argumento deve ser um número. Ex -> ./netifstat.sh 10"
    usage # Menu de execução do programa.
    exit 1 # Terminar o programa
fi
```

Img 5 – Verificação do último argumento

Caso não se cumpra alguma das condições, em cada uma delas será chamada a função *usage* e o programa irá terminar.

2.2.3 – While getopt

No decorrer do seguinte ciclo *while*, utilizámos o comando *getopts* que nos permitiu passar várias opções e fazer o tratamento das mesmas.

```
# While para tratamento das opções seleccionadas.
while getopts "c:bkmp:trTRv1" option; do
```

Img 6 – while getopt para as opções

Como se pode observar em cima, utiliza-se um ciclo *while* e é dentro deste ciclo que se vai fazer todo o tratamento, verificação e validação das opções seleccionadas. No comando *getopts* foram passadas várias opções e podemos verificar a presença dos ":" à frente de daquelas que necessitam de um determinado argumento adicional.

Agora dentro do comando *getopts*, prosseguimos à adição das opções passadas e dos seus respetivos argumentos.

```
#Adicionar ao array optsOrd as opcoes passadas ao correr o programa.
if [[ -z "$OPTARG" ]]; then
    optsOrd[$option]="blank" # Caso a opção não precise de argumento, passa blank para o array. Ex: -b -> blank
else
    optsOrd[$option]="${OPTARG}" # Caso precisem de argumento, guarda o argumento no array.
fi
```

Img 7 – Adição das opções ao array associativo

Caso a opção não precise de argumento, passa "blank" para o array entrando como *value* associado à *key* correspondente à opção. Por outro lado, se a mesma tiver argumento, a *key* guardada no array vai ser a mesma a opção passada, porém, o seu *value*, desta vez, vai ser o argumento.

De seguida, utilizámos a instrução bash *case* de forma a simplificar as expressões condicionais, visto que temos bastantes opções diferentes.

No caso da opção -c, em que é preciso um argumento, guardamos o respetivo *value* na variável opção passada na variável global *\$c*.

```
case $option in
c) #Seleção das interfaces a visualizar através de uma expressão regular.
    c=${optsOrd[c]}
    let "ctr+=2" # Acrescentar 2 ao valor de controlo dos argumentos.
;;
```

Img 8 – método case

Para a opção -p o processo é semelhante, com a diferença que, este caso há necessidade de usar uma condição onde se verifica se o *value* é ou não um número positivo.


```
p) #Seleção do número de interfaces de redes a visualizar.
p=${optsOrd[p]}
if ! [[ $p =~ $nre ]]; then
    echo "Error : A opção -p requer que se indique o número de redes a visualizar. Ex -> netifstat -p 2 10" >&2
    usage # Menu de execução do programa.
    exit 1 # Terminar o programa
fi
let "ctr+=2" # Acrescentar 2 ao valor de controlo dos argumentos.
;;
```

Img 9 – Tratamento da opção -p

Agora passando para as opções onde não são necessários argumentos. Na opção -l, apenas será mudado o valor da variável global *\$l* para 1 (variável responsável para transportar o valor do loop de -l).

```
l) #Seleção do intervalo de tempo entre execuções do loop.
l=1
let "ctr+=1" # Acrescentar 1 ao valor de controlo dos argumentos.
;;
```

Img 10 – Tratamento da opção -l

No caso das opções -b, -k e -m, existe uma condição para verificar que o utilizador apenas introduz uma dessas opções, utilizando a variável global *\$i*. De seguida, outras duas condições onde, caso a opção selecionada por o utilizador for -k a unidade de visualização vai ser alterada para kilobytes e outra que, caso a opção for -m a unidade de visualização vai ser em megabytes. Em ambas, o valor da variável global *\$d* vai ser alterado para 1 ou para 2, respetivamente. Variável esta que depois vai ser responsável pela conversão dos valores para as unidades desejadas pelo utilizador.

```
b | k | m ) # Mudar a unidade de visualização (Bytes, Kilobytes, Megabytes).
if [[ $i = 1 ]]; then
    echo "Só é permitido o uso de uma das opções : -b, -k ou -m. Ex -> ./netifstat -b 10"
    usage # Menu de execução do programa.
    exit 1 # Terminar o programa
fi
i=1
if [[ ${optsOrd[k]} == "blank" ]]; then # Mudar a unidade de visualização para kilobytes.
    d=1;
fi
if [[ ${optsOrd[m]} == "blank" ]]; then # Mudar a unidade de visualização para megabytes.
    d=2;
fi
let "ctr+=1" # Acrescentar 1 ao valor de controlo dos argumentos.
;;
```

Img 11 – Tratamento das opções -b, -k e -m

Já nas opções -t, -r, -T e -R, tal como nas anteriores, necessitámos de confirmar que o utilizador apenas selecionou uma delas e para tal utilizamos a variável *\$m*. Alterámos também o valor da variável *\$reverse* para "r" de forma que a ordenação pré-definida por estas opções seja a forma decrescente. Seguidamente, criámos várias condições *if* onde, dependendo da opção selecionada, o valor da variável global *\$k*, inicialmente a 1, vai ser alterado para o número da coluna que o utilizador pretende ordenar.


```

t | r | T | R) # Ordenação da tabela por coluna (decrescente).
reverse="r"
if [[ $m = 1 ]]; then
    echo "Só é permitido o uso de uma das opções : -t, -r, -T ou -R. Ex -> ./netifstat -r 10"
    usage # Menu de execução do programa.
    exit 1
fi
let "m+=1"
if [[ $option == "t" ]]; then # Uso da opção -t.
    k=2 # Alterar a coluna 2 da impressão. Coluna dos valores de TX.
fi
if [[ $option == "r" ]]; then # Uso da opção -r.
    k=3 # Alterar a coluna 3 da impressão. Coluna dos valores de RX.
fi
if [[ $option == "T" ]]; then # Uso da opção -T.
    k=4 # Alterar a coluna 4 da impressão. Coluna dos valores de TRATE.
fi
if [[ $option == "R" ]]; then # Uso da opção -R.
    k=5 # Alterar a coluna 5 da impressão. Coluna dos valores de RRATE.
fi
let "ctr+=1" # Acrescentar 1 ao valor de controlo dos argumentos.
;;

```

Img 12 – Tratamento das opções -t, -r, -T e -R

Ainda no *case*, passamos para o caso da opção -v, onde se criou mais uma condição, desta vez para verificar o valor da variável global `$reverse`. Como a opção -v é verificada a seguir às opções -t, -r, -T e -R, então é possível saber se essas opções alteraram a variável `$reverse` para "r" e, portanto, a variável será novamente alterada para "". Caso contrário, a variável é alterada para "r" de forma a imprimir a primeira coluna (coluna com o nome das interfaces de rede) pela ordem alfabética reversa.

```

v) # Ordenação reversa (crescente).
if [[ $reverse == "r" ]]; then # Caso o $reverse já tenha sido mudado em "t | r | T | R)"
    reverse="" # Fazer a tabela imprimir de forma crescente
else
    reverse="r"
fi
let "ctr+=1" # Acrescentar 1 ao valor de controlo dos argumentos.
;;

```

Img 13 – Tratamento da opção -v

Por último, temos ainda o caso "*" que controla a passagem de argumentos inválidos. Caso seja passada uma qualquer opção não predefinida no programa (ex: `./netifstat -h 10`), o programa imprime uma mensagem a avisar do erro, executa a função `usage()` e termina.

```

*) # Uso de argumentos inválidos.
echo "Uso de argumentos inválidos."
usage # Menu de execução do programa.
exit 1 # Terminar o programa
;;

```

Img 14 – Tratamento de opções inválidas

2.2.4 – Variável *\$ctr* e Controlo do Número de Argumentos

Como se pode notar, em cada uma das condições programadas anteriormente o valor da variável global *\$ctr* (inicializada a 1) foi sendo incrementada. No caso das opções -c e -p este valor aumentava 2, pois corresponde ao número de argumentos passados em cada uma destas opções (opção + argumento adicional). Já no caso das opções restantes, o valor apenas aumentava 1. Esta variável no fim vai possuir o valor total de argumentos passados. De modo a evitar casos em que o programa corre se forem usados argumentos do tipo `./netifstat.sh -c 2`, criámos uma condição onde se vai averiguar se o número de argumentos passados é igual ou não, ao número de argumentos contados e validados pela variável *\$ctr*. Caso sejam diferentes, o programa não corre e termina.

```
# Verificar se o valor do controlo de argumentos é igual ao número de argumentos passados.
# Evitar casos em que o programa corre se forem usados argumentos do tipo -> ./netifstat -c 2
if ! [[ $# == $ctr ]]; then
    echo "Uso de argumentos inválidos. Poucos argumentos foram passados."
    usage # Menu de execução do programa.
    exit 1 # Terminar o programa
fi
```

Img 15 – Verificação da variável *\$ctr*

Por exemplo, caso seja executado o programa com o comando `./netifstat.sh -c 2`, o valor de *\$ctr* será igual a 3 – inicialização a 1 + 2 pela passagem no caso -c –, no entanto o número de argumentos passados (*\$#*) é igual a 2 – `./netifstat.sh -c 2`.

2.2.5 – Função *getTable* Dependendo da Opção -l

Em último lugar e ainda acerca do tratamento das opções, criámos outro *if* onde, caso a opção -l tenha sido passada como argumento e o seu valor tenha sido alterado para 1, ou seja, maior que 0, vai ser executado um loop infinito da função *getTable*. Caso a opção não tenha sido passada, a função é executada apenas uma única vez.

```
# Execução da função getTable dependendo da opção -l (loop)
if [[ $l -gt 0 ]]; then
    while true; do # Loop sem quebras.
        getTable
        echo
    done
else
    getTable # Caso em que não se passa o argumento -l.
fi
```

Img 16 – Tipos de execução

3 – Processos de Execução

3.1 – Função getTable

A função *getTable* é a função principal do *script*. É a partir desta função que, após terem sido lidos os argumentos passados, serão obtidos os dados necessários e é impressa a tabela com os valores desejados. A função é composta maioritariamente por três ciclos *for*.

Apesar de os três ciclos *for* terem as mesmas condições, foi escolhido este método de trabalho, devido a tornar a execução do programa mais rápida e mais próxima do que é desejado, dado que, só precisamos de fazer *sleep* uma vez. Os dois primeiros ciclos *for* são separados por esse *sleep*.

```
sleep $t # Tempo de espera entre pedidos da quantidade de dados transmitidos e recebidos. Passado como último argumento.
```

Img 17 – Sleep entre os ciclos *for*

3.1.1 – Condições Dos Ciclos For

Como já tinha sido referido anteriormente, todos os ciclos *for* têm as mesmas condições.

```
for net in /sys/class/net/[[:alnum:]]*; do # Procurar por todas as interfaces de rede disponíveis.
    if [[ -r $net/statistics ]]; then
        f="$(basename -- $net)" # Passar $f com o nome da interface de rede.
```

Img 18 – Primeiras condições dos ciclos *for*

Em cada execução do ciclo, guarda-se na variável local *\$net* os dados do diretório de uma interface de rede. De seguida aplica-se uma condição para verificar se temos permissões nas *statistics* da interface de rede que cada ciclo *for* estiver a iterar. Após isso, é guardado o nome da interface de rede na variável local *\$f*, para posteriormente se poder usar nos arrays associativos.

```
tsora@DESKTOP-RRHS0DA:/mnt/c/WINDOWS/system32$ ls /sys/class/net
bond0 bonding_masters dummy0 eth0 lo sit0 tunl0
tsora@DESKTOP-RRHS0DA:/mnt/c/WINDOWS/system32$ ls /sys/class/net/lo/statistics
collisions rx_crc_errors rx_frame_errors rx_over_errors tx_carrier_errors tx_fifo_errors
multicast rx_dropped rx_length_errors rx_packets tx_compressed tx_heartbeat_errors
rx_bytes rx_errors rx_missed_errors tx_aborted_errors tx_dropped tx_packets
rx_compressed rx_fifo_errors rx_nohandler tx_bytes tx_errors tx_window_errors
tsora@DESKTOP-RRHS0DA:/mnt/c/WINDOWS/system32$
```

Img 19 – Alguns exemplos dos diretórios disponíveis para a variável *\$net*

Para além das condições iniciais, todos os ciclos *for* apenas irão trabalhar com as interfaces de rede que o utilizador identificou ao passar uma expressão regular com a opção *-c*. Caso esta opção tenha sido passada, todas as interfaces de rede que não foram identificadas na expressão regular, serão ignoradas e impressões também não serão feitas.

```
# Condição para apenas trabalhar com interfaces de rede que coincidam com a expressão regular passada pela opção -c.
if [[ -v optsOrd[c] && ! $f =~ ${optsOrd[c]} ]]; then
    continue
fi
```

Img 20 – Condição que trata do valor da opção *-c*

3.1.2 – Primeiro Ciclo For (Primeiro pedido da quantidade de dados)

No primeiro ciclo *for* tem como principal função obter a quantidade de dados transmitidos e recebidos nas interfaces de rede disponíveis aquando da execução do programa.

```
function getTable() { # Função principal do programa. Obtém os valores desejados, ordena-os e imprimi-los.
for net in /sys/class/net/[[:alnum:]]*; do # Procurar por todas as interfaces de rede disponíveis.
if [[ -r $net/statistics ]]; then
f="$($basename -- $net)" # Passar $f com o nome da interface de rede.
# Condição para apenas trabalhar com interfaces de rede que coincidam com a expressão regular passada pela opção -c.
if [[ -v optsOrd[c] && ! $f =~ ${optsOrd[c]} ]]; then
continue
fi
if [[ -z ${rx1[$f]} ]]; then # Caso em que o valor de RX1 ainda não está definido.
rx1[$f]=$(cat $net/statistics/rx_bytes | grep -o -E '[0-9]+') # Obter do valor de RX1 em bytes, na primeira execução.
else
rx1[$f]=${rx2[$f]} # Obter do valor de RX1 em bytes, a partir do RX2 da execução anterior.
fi
if [[ -z ${tx1[$f]} ]]; then # Caso em que o valor de TX1 ainda não está definido.
tx1[$f]=$(cat $net/statistics/tx_bytes | grep -o -E '[0-9]+') # Obter do valor de TX1 em bytes, na primeira execução.
else
tx1[$f]=${tx2[$f]} # Obter do valor de TX1 em bytes, a partir do TX2 da execução anterior.
fi
fi
done
```

Img 21 – Primeiro ciclo *for*

O terceiro e quarto *if* serem para verificar se os arrays associativos *\$rx1* e *\$tx1* de cada interface já foram inicializados. Caso ainda não tenham sido, utiliza-se um *cat* e um *grep* de forma a retornar e guardar os valores que queremos obter dos diretórios *rx_bytes* e *tx_bytes*. Caso esses valores já existam de uma execução anterior da função – caso que se aplica apenas se se passar a opção *-l* ao executar o programa – então esses valores são substituídos pelos valores de *\$rx2* e *\$tx2* (valores que são posteriormente obtidos no segundo ciclo *for*).

3.1.3 – Segundo Ciclo For (Cálculo dos valores necessários à impressão)

Após ter o tempo de espera do *sleep*, é executado o segundo ciclo *for* da função *getTable*. Assim como o primeiro ciclo, este ciclo *for* obtém, mais uma vez, a quantidade de dados transmitidos e recebidos nas interfaces de rede disponíveis.

```
for net in /sys/class/net/[[:alnum:]]*; do # Procurar por todas as interfaces de rede disponíveis.
if [[ -r $net/statistics ]]; then
f="$($basename -- $net)" # Passar $f com o nome da interface de rede.
# Condição para apenas trabalhar com interfaces de rede que coincidam com a expressão regular passada pela opção -c.
if [[ -v optsOrd[c] && ! $f =~ ${optsOrd[c]} ]]; then
continue
fi
rx2[$f]=$(cat $net/statistics/rx_bytes | grep -o -E '[0-9]+') # Obter do valor de RX2 em bytes.
tx2[$f]=$(cat $net/statistics/tx_bytes | grep -o -E '[0-9]+') # Obter do valor de TX2 em bytes.
rxb=$((rx2[$f] - rx1[$f])) # Obter do valor de RX em bytes, subtraindo RX2 por RX1.
txb=$((tx2[$f] - tx1[$f])) # Obter do valor de TX em bytes, subtraindo TX2 por TX1.
rrate=$(bc <<< "scale=3;$rxb/$t") # Obter do valor de RRATE em bytes.
trate=$(bc <<< "scale=3;$txb/$t") # Obter do valor de TRATE em bytes.
mult=$((1024 * d)) # Calculo usado para alterar a unidade desejada (Bytes, Kilobytes, Megabytes).
rx[$f]=$(bc <<< "scale=3;$rxb/$mult") # Alterar RX para unidade desejada e salva-la no array.
tx[$f]=$(bc <<< "scale=3;$txb/$mult") # Alterar TX para unidade desejada e salva-la no array.
rrate[$f]=$(bc <<< "scale=3;$rrate/$mult") # Alterar RRATE para unidade desejada e salva-la no array.
trate[$f]=$(bc <<< "scale=3;$trate/$mult") # Alterar TRATE para unidade desejada e salva-la no array.
if [[ -z ${txtot[$f]} ]]; then # Inicialização do TXTOT se ele ainda não existir.
txtot[$f]=0
fi
if [[ -z ${rxtot[$f]} ]]; then # Inicialização do RXTOT se ele ainda não existir.
rxtot[$f]=0
fi
txtot[$f]=$(bc <<< "scale=3;${txtot[$f]}+${tx[$f]}") # Soma do valor de TX anterior ao TX total.
rxtot[$f]=$(bc <<< "scale=3;${rxtot[$f]}+${rx[$f]}") # Soma do valor de RX anterior ao RX total.
fi
done
```

Img 22 – Segundo ciclo *for*

Mais uma vez, o valor da quantidade de dados transmitidos e recebidos nas interfaces de rede é registado nos arrays associativos, neste caso, a *\$rx2* e *\$tx2*. Após termos os valores dessas variáveis e os valores de *\$rx1* e *\$tx1* obtidos no ciclo *for* anterior, então fazemos uma subtração, calculando assim a quantidade de dados transmitidos e recebido durante o tempo do *sleep* e guardando esses valores nas variáveis *\$rx* e *\$tx*.

Após se obter os valores de RX e TX de cada uma das interfaces de rede, obtém-se o RRATE e TRATE, dividindo os valores de *\$rx* e *\$tx* pelo tempo de espera do sleep, no caso *\$t*, e guarda-se esses valores nas variáveis locais *\$rrate* e *\$trate*.

Como os valores até agora obtidos são em bytes, utilizamos a variável local *\$mult* para ajudar na transformação para kilobytes ou megabytes. O valor de *\$mult* é obtido através de um calculo que depende do valor da variável global *\$d*. Caso *\$d* seja igual a 0 (valor com que é inicializada), então *\$mult* também é igual a 0. Caso, nas opções passadas, se tenha usado -k ou -m, o valor de *\$d* passa a ser 1 ou 2 e o valor de *\$mult* passa a ser 1024¹ e 1024², respetivamente.

```
b | k | m ) # Mudar a unidade de visualização (Bytes, Kilobytes, Megabytes).
if [[ $i = 1 ]]; then
    echo "Só é permitido o uso de uma das opções : -b, -k ou -m. Ex -> ./netifstat -b 10"
    usage # Menu de execução do programa.
    exit 1 # Terminar o programa
fi
i=1
if [[ ${optsOrd[k]} == "blank" ]]; then # Mudar a unidade de visualização para kilobytes.
    d=1;
fi
if [[ ${optsOrd[m]} == "blank" ]]; then # Mudar a unidade de visualização para megabytes.
    d=2;
fi
let "ctr+=1" # Acrescentar 1 ao valor de controlo dos argumentos.
;;
```

Img 23 – Alteração da variável global *\$d*

Dividindo cada um dos valores das variáveis *\$rx*, *\$tx*, *\$rrate* e *\$trate* por *\$mult*, obtém-se esses valores na unidade de visualização e então eles podem ser guardados nos arrays associativos *\$rx*, *\$tx*, *\$rrate* e *\$trate*.

Ainda no segundo ciclo for, faz-se o cálculo dos valores de RX total e TX total. Valores que são inicializados a 0, caso ainda não tenham sido inicializados, e então incrementados com o valor de *\$rx* e *\$tx* da interface de rede e guardados nos arrays associativos *\$rxtot* e *\$txtot*.

```
if [[ -z ${txtot[$f]} ]]; then # Inicialização do TXTOT se ele ainda não existir.
    txtot[$f]=0
fi
if [[ -z ${rxtot[$f]} ]]; then # Inicialização do RXTOT se ele ainda não existir.
    rxtot[$f]=0
fi
txtot[$f]=$((bc <<< "scale=3;${txtot[$f]}+${tx[$f]}")) # Soma do valor de TX anterior ao TX total.
rxtot[$f]=$((bc <<< "scale=3;${rxtot[$f]}+${rx[$f]}")) # Soma do valor de RX anterior ao RX total.
fi
```

Img 24 – Cálculo do RX total e TX total

4 – Impressão dos dados

4.1 – Impressão do cabeçalho da tabela

Antes do terceiro ciclo `for`, existe uma condição que possibilita a impressão do cabeçalho da tabela, dependendo da opção `-l`. Caso a opção `-l` tenha sido passada na execução do programa, então a variável global `$l` será igual a 1 e o cabeçalho da tabela incluirá “TXTOT” e “RXTOT” na impressão.

```
if [[ $l == 0 ]]; then # Caso em que não se passou a opção -l e não é preciso calcular o RX e TX total.
    printf "%-15s %-15s %-15s %-15s %-15s\n" "NETIF" "TX" "RX" "TRATE" "RRATE" # Imprimir o cabeçalho da tabela.
else
    printf "%-15s %-15s %-15s %-15s %-15s %-15s %-15s %-15s\n" "NETIF" "TX" "RX" "TRATE" "RRATE" "TXTOT" "RXTOT" # Imprimir o cabeçalho da tabela.
fi
```

Img 25 – Tipos de impressão de cabeçalho

4.2 – Impressão dos dados das interfaces de rede

O terceiro ciclo `for` é o ciclo responsável pela impressão dos dados que o utilizador irá ver. Este ciclo `for` funciona a partir dos dados previamente calculados na função `getTable` e de condições que definem os lados a imprimir.

```
n=0 # Usada para controlar o número de interfaces de rede impressas.
for net in $(cat /sys/class/net/[[!alnum:]]); do # Procurar por todas as interfaces de rede disponíveis.
    if [[ -r $net/statistics ]]; then
        fe=$(basename -- $net) # Passar $f com o nome da interface de rede.
        if [[ $n -lt $p || $p = -1 ]]; then # Condição para apenas serem vistos o número de interfaces passados pela opção -p.
            # Condição para apenas trabalhar com interfaces de rede que coincidam com a expressão regular passada pela opção -c.
            if [[ -v optsOrd[c] && ! $f =~ ${optsOrd[c]} ]]; then
                continue
            fi
            if [[ $l == 0 ]]; then # Caso em que não se passou a opção -l e não é preciso calcular o RX e TX total.
                printf "%-15s %-15s %-15s %-15s\n" "$f" "${tx[$f]}" "${rx[$f]}" "${trate[$f]}" "${rrate[$f]}" # Imprimir os valores da tabela.
            else
                printf "%-15s %-15s %-15s %-15s %-15s %-15s %-15s %-15s\n" "$f" "${tx[$f]}" "${rx[$f]}" "${trate[$f]}" "${rrate[$f]}" "${txtot[$f]}" "${rxtot[$f]}" # Imprimir os valores da tabela.
            fi
            let "n+=1" # Incrementar o valor de n.
        fi
    fi
done | sort -k$k$reverse # Ordenar o output da tabela a partir da coluna ( $k ), decrescente ou crescente ( $reverse ).
```

Img 26 – Tipos dos dados das interfaces de rede

Ainda antes do ciclo `for`, é inicializada a variável `$n`. Esta variável, que posteriormente será incrementada em cada iteração do ciclo, é uma variável que conta o número de interfaces de rede que já foram impressas. Caso essa variável seja superior ao valor de `$p` – variável inicializado a -1, e alterado caso seja passado o argumento `-p` para o valor desejado – então, a partir desse ponto, não serão impressas mais interfaces.

De seguida, aparece uma condição semelhante à condição da impressão do cabeçalho da tabela, mas desta vez os `printf` utilizam os valores de que se obtiveram nos ciclos `for` anteriores - `$tx`, `$rx`, `$trate`, `$rrate`, (`$txtot` e `$rxtot` no uso da opção `-l`) de cada uma das interfaces a ser impressas.

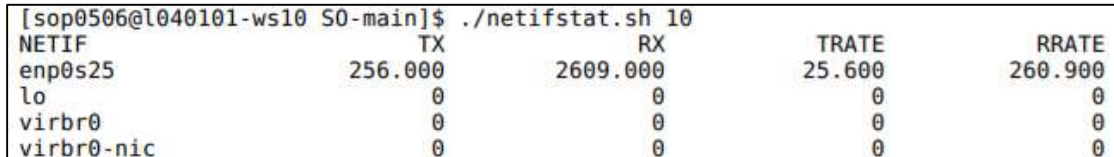
Para além de todas estas funcionalidades, o terceiro ciclo `for` ainda dá `sort` à tabela os valores das interfaces de rede impressas. O `sort` funciona partir `$k`, que determina a coluna a ser ordenada e do `$reverse`, que define se a tabela vai ser ordenada de forma decrescente ou crescente. As formas como as variáveis globais `$k` e `$reverse` funcionam estão explicadas no tópico 2.2.3.

5 – Resultados

5.1 – Resultados Válidos

De forma a mostrar algumas funcionalidades, realizámos vários testes, possivelmente uteis para um utilizador do programa.

No primeiro teste, é apenas passado o último argumento “10”.

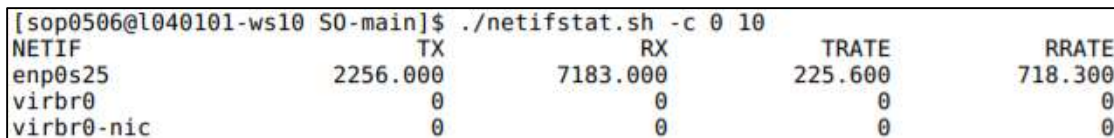


```
[sop0506@l040101-ws10 S0-main]$ ./netifstat.sh 10
```

NETIF	TX	RX	TRATE	RRATE
enp0s25	256.000	2609.000	25.600	260.900
lo	0	0	0	0
virbr0	0	0	0	0
virbr0-nic	0	0	0	0

Img 27 – ./netifstat.sh 10

Seguidamente, de forma a testar a passagem da opção “-c”, realizámos um teste em que pedimos todas interfaces de rede que correspondessem à expressão regular “0”.

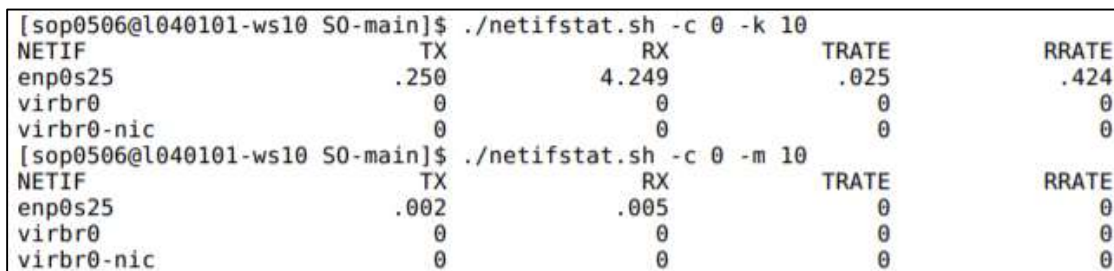


```
[sop0506@l040101-ws10 S0-main]$ ./netifstat.sh -c 0 10
```

NETIF	TX	RX	TRATE	RRATE
enp0s25	2256.000	7183.000	225.600	718.300
virbr0	0	0	0	0
virbr0-nic	0	0	0	0

Img 28 – ./netifstat.sh -c 0 10

Depois, verificámos a eficácia da transformação das unidades de visualização através da passagem das opções -k e -m (-b não iria mostrar alterações já que é a unidade predefinida).



```
[sop0506@l040101-ws10 S0-main]$ ./netifstat.sh -c 0 -k 10
```

NETIF	TX	RX	TRATE	RRATE
enp0s25	.250	4.249	.025	.424
virbr0	0	0	0	0
virbr0-nic	0	0	0	0

```
[sop0506@l040101-ws10 S0-main]$ ./netifstat.sh -c 0 -m 10
```

NETIF	TX	RX	TRATE	RRATE
enp0s25	.002	.005	0	0
virbr0	0	0	0	0
virbr0-nic	0	0	0	0

Img 29 – ./netifstat.sh -c 0 -k 10

Agora de forma a testar a passagem da opção -p, executámos o código 3 vezes, passando diferentes números de interfaces a visualizar.

```
[sop0506@l040101-ws10 S0-main]$ ./netifstat.sh -p 1 10
```

NETIF	TX	RX	TRATE	RRATE
enp0s25	8701.000	10563.000	870.100	1056.300

```
[sop0506@l040101-ws10 S0-main]$ ./netifstat.sh -p 2 10
```

NETIF	TX	RX	TRATE	RRATE
enp0s25	2565.000	5555.000	256.500	555.500
lo	0	0	0	0

```
[sop0506@l040101-ws10 S0-main]$ ./netifstat.sh -p 3 10
```

NETIF	TX	RX	TRATE	RRATE
enp0s25	442.000	5724.000	44.200	572.400
lo	0	0	0	0
virbr0	0	0	0	0

Img 30 – ./netifstat.sh -p (n) 10

O programa também possibilita a ordenação da tabela por qualquer uma das colunas, passando os argumentos -t, -r, -T ou -R.

```
tsora@DESKTOP-RRHS0DA:/mnt/c/Users/tiago/Documents/GitHub/SO$ ./netifstat.sh -r 20
```

NETIF	TX	RX	TRATE	RRATE
eth0	0	926.000	0	46.300
bond0	0	0	0	0
dummy0	0	0	0	0
lo	0	0	0	0
sit0	0	0	0	0
tunl0	0	0	0	0

Img 31 – ./netifstat.sh -r 10

Para verificar a ordenação inversa, também executámos programa passando lhe a opção -v.

```
[sop0506@l040101-ws10 S0-main]$ ./netifstat.sh -v 10
```

NETIF	TX	RX	TRATE	RRATE
virbr0-nic	0	0	0	0
virbr0	0	0	0	0
lo	0	0	0	0
enp0s25	0	3717.000	0	371.700

Img 32 – ./netifstat.sh -v 10

Para terminar a verificação de todas as funcionalidades, adicionámos ao comando anterior a opção -l, o que nos permitiu assistir à execução do programa em loop.

```
[sop0506@l040101-ws10 S0-main]$ ./netifstat.sh -v -l 10
```

NETIF	TX	RX	TRATE	RRATE	TXTOT	RXTOT
virbr0-nic	0	0	0	0	0	0
virbr0	0	0	0	0	0	0
lo	0	0	0	0	0	0
enp0s25	372.000	1892.000	37.200	189.200	372.000	1892.000

NETIF	TX	RX	TRATE	RRATE	TXTOT	RXTOT
virbr0-nic	0	0	0	0	0	0
virbr0	0	0	0	0	0	0
lo	0	0	0	0	0	0
enp0s25	4593.000	8678.000	459.300	867.800	4965.000	10570.000

NETIF	TX	RX	TRATE	RRATE	TXTOT	RXTOT
virbr0-nic	0	0	0	0	0	0
virbr0	0	0	0	0	0	0
lo	0	0	0	0	0	0
enp0s25	0	2866.000	0	286.600	4965.000	13436.000

Img 33 – ./netifstat.sh -v -l 10

Para terminar os testes válidos corremos o programa com o máximo de argumentos possíveis usando as opções -c, -k, -p, -R, -v e -l ao mesmo tempo e o resultado foi o esperado.

```
[sop0506@l040101-ws10 S0-main]$ ./netifstat.sh -c 0 -k -p 2 -R -v -l 10
```

NETIF	TX	RX	TRATE	RRATE	TXTOT	RXTOT
virbr0	0	0	0	0	0	0
enp0s25	.068	3.250	.006	.325	.068	3.250
NETIF	TX	RX	TRATE	RRATE	TXTOT	RXTOT
virbr0	0	0	0	0	0	0
enp0s25	1.297	5.082	.129	.508	1.365	8.332
NETIF	TX	RX	TRATE	RRATE	TXTOT	RXTOT
virbr0	0	0	0	0	0	0
enp0s25	.130	2.964	.013	.296	1.495	11.296
NETIF	TX	RX	TRATE	RRATE	TXTOT	RXTOT
virbr0	0	0	0	0	0	0
enp0s25	.250	4.226	.025	.422	1.745	15.522
NETIF	TX	RX	TRATE	RRATE	TXTOT	RXTOT
virbr0	0	0	0	0	0	0
enp0s25	1.736	4.875	.173	.487	3.481	20.397
NETIF	TX	RX	TRATE	RRATE	TXTOT	RXTOT
virbr0	0	0	0	0	0	0
enp0s25	0	3.163	0	.316	3.481	23.560
NETIF	TX	RX	TRATE	RRATE	TXTOT	RXTOT
virbr0	0	0	0	0	0	0
enp0s25	.136	3.606	.013	.360	3.617	27.166
NETIF	TX	RX	TRATE	RRATE	TXTOT	RXTOT
virbr0	0	0	0	0	0	0
enp0s25	.363	2.613	.036	.261	3.980	29.779
NETIF	TX	RX	TRATE	RRATE	TXTOT	RXTOT
virbr0	0	0	0	0	0	0
enp0s25	2.887	5.512	.288	.551	6.867	35.291

Imagem 34 – ./netifstat.sh -c 0 -k -p 2 -R -v -l 10

5.2 – Resultados Inválidos

Como já tinha sido dito no relatório, o programa está preparado para tratar a maioria (se não todos) dos possíveis erros que possam ser cometidos pelo utilizador ao executar o programa. Através de um conjunto de verificações anteriormente explicadas podemos ter diferentes tipos de mensagens de erros. Sempre que o utilizador erra no uso do programa, o programa corre a função *usage* e termina.

Neste primeiro caso, não foi passado o último argumento (número inteiro no final do comando), pelo que o programa mostrou ao utilizador uma mensagem a lembrar dessa necessidade.

```
tsora@DESKTOP-RRHS0DA:/mnt/c/Users/tiago/Documents/GitHub/SO$ ./netifstat.sh
Necessário, pelo menos, o período de tempo desejado (segundos). Ex -> ./netifstat.sh 10
Menu de Uso e Execução do Programa.
-c [NETIF] : Seleção das interfaces de rede, [NETIF], a visualizar através de uma expressão regular.
-b          : Visualização das quantidades em bytes.
-k          : Visualização das quantidades em kilobytes.
-m          : Visualização das quantidades em megabytes.
-p [n]      : Número, [n], de interfaces de redes a visualizar.
-t          : Ordenação da tabela por TX (decrescente).
-r          : Ordenação da tabela por RX (decrescente)..
-T          : Ordenação da tabela por TRATE (decrescente).
-R          : Ordenação da tabela por RRATE (decrescente).
-v          : Ordenação reversa (crescente).
-l          : Loop de execução do programa a cada [s] segundos.
ALERTAS -> As opções -t,-r,-T,-R não podem ser utilizadas em simultâneo.
           O último argumento passado tem de o período de tempo desejado (segundos).
```

Imagem 35 – ./netifstat.sh

No entanto mesmo que, o último argumento seja realmente passado, apenas é aceite se for um número inteiro positivo, pois caso contrário o programa irá executar outra mensagem de erro.

```
tsora@DESKTOP-RRHS0DA:/mnt/c/Users/tiago/Documents/GitHub/SO$ ./netifstat.sh a
O último argumento deve ser um número. Ex -> ./netifstat.sh 10
Menu de Uso e Execução do Programa.
  -c [NETIF] : Seleção das interfaces de rede, [NETIF], a visualizar através de uma expressão regular.
  -b          : Visualização das quantidades em bytes.
  -k          : Visualização das quantidades em kilobytes.
  -m          : Visualização das quantidades em megabytes.
  -p [n]      : Número, [n], de interfaces de redes a visualizar.
  -t          : Ordenação da tabela por TX (decrescente).
  -r          : Ordenação da tabela por RX (decrescente)..
  -T          : Ordenação da tabela por TRATE (decrescente).
  -R          : Ordenação da tabela por RRATE (decrescente).
  -v          : Ordenação reversa (crescente).
  -l          : Loop de execução do programa a cada [s] segundos.
ALERTAS -> As opções -t,-r,-T,-R não podem ser utilizadas em simultâneo.
           O último argumento passado tem de o período de tempo desejado (segundos).
```

Img 36 – ./netifstat.sh a

A única opção que necessita de um argumento adicional e que pode causar erros no programa é a opção -p, pois a seguir a esta deve sempre se encontrar um número inteiro positivo. Caso o utilizador tente passar uma *string* que não corresponda à necessidade, o programa irá reconhecer esse erro.

```
tsora@DESKTOP-RRHS0DA:/mnt/c/Users/tiago/Documents/GitHub/SO$ ./netifstat.sh -p abc 10
Error : A opção -p requer que se indique o número de redes a visualizar. Ex -> netifstat -p 2 10
Menu de Uso e Execução do Programa.
  -c [NETIF] : Seleção das interfaces de rede, [NETIF], a visualizar através de uma expressão regular.
  -b          : Visualização das quantidades em bytes.
  -k          : Visualização das quantidades em kilobytes.
  -m          : Visualização das quantidades em megabytes.
  -p [n]      : Número, [n], de interfaces de redes a visualizar.
  -t          : Ordenação da tabela por TX (decrescente).
  -r          : Ordenação da tabela por RX (decrescente)..
  -T          : Ordenação da tabela por TRATE (decrescente).
  -R          : Ordenação da tabela por RRATE (decrescente).
  -v          : Ordenação reversa (crescente).
  -l          : Loop de execução do programa a cada [s] segundos.
ALERTAS -> As opções -t,-r,-T,-R não podem ser utilizadas em simultâneo.
           O último argumento passado tem de o período de tempo desejado (segundos).
```

Img 37 – ./netifstat.sh -p abc 10

Ainda, no decorrer do tratamento de opções, existem dois grupos de opções em podem causar erros no programa. Como havia sido dito antes, o utilizador só pode utilizar uma das opções entre -b, -k e -m e entre -t, -r, -T e -R. Caso contrário o programa mandará uma das seguintes mensagens de erro no terminal.

```
tsora@DESKTOP-RRHS0DA:/mnt/c/Users/tiago/Documents/GitHub/SO$ ./netifstat.sh -b -k 10
Só é permitido o uso de uma das opções : -b, -k ou -m. Ex -> ./netifstat -b 10
Menu de Uso e Execução do Programa.
-c [NETIF] : Seleção das interfaces de rede, [NETIF], a visualizar através de uma expressão regular.
-b          : Visualização das quantidades em bytes.
-k          : Visualização das quantidades em kilobytes.
-m          : Visualização das quantidades em megabytes.
-p [n]      : Número, [n], de interfaces de redes a visualizar.
-t          : Ordenação da tabela por TX (decrescente).
-r          : Ordenação da tabela por RX (decrescente)..
-T          : Ordenação da tabela por TRATE (decrescente).
-R          : Ordenação da tabela por RRATE (decrescente).
-v          : Ordenação reversa (crescente).
-l          : Loop de execução do programa a cada [s] segundos.
ALERTAS -> As opções -t,-r,-T,-R não podem ser utilizadas em simultâneo.
           O último argumento passado tem de o período de tempo desejado (segundos).
```

Img 38 – ./netifstat.sh -b -k 10

```
tsora@DESKTOP-RRHS0DA:/mnt/c/Users/tiago/Documents/GitHub/SO$ ./netifstat.sh -t -R 10
Só é permitido o uso de uma das opções : -t, -r, -T ou -R. Ex -> ./netifstat -r 10
Menu de Uso e Execução do Programa.
-c [NETIF] : Seleção das interfaces de rede, [NETIF], a visualizar através de uma expressão regular.
-b          : Visualização das quantidades em bytes.
-k          : Visualização das quantidades em kilobytes.
-m          : Visualização das quantidades em megabytes.
-p [n]      : Número, [n], de interfaces de redes a visualizar.
-t          : Ordenação da tabela por TX (decrescente).
-r          : Ordenação da tabela por RX (decrescente)..
-T          : Ordenação da tabela por TRATE (decrescente).
-R          : Ordenação da tabela por RRATE (decrescente).
-v          : Ordenação reversa (crescente).
-l          : Loop de execução do programa a cada [s] segundos.
ALERTAS -> As opções -t,-r,-T,-R não podem ser utilizadas em simultâneo.
           O último argumento passado tem de o período de tempo desejado (segundos).
```

Img 39 – ./netifstat.sh -t -R 10

Para terminal a validação das opções passadas, o programa têm ainda uma mensagem de erro para caso de o utilizador executar o programa com uma opção que não contes da lista das programadas. Todas as opções, iniciadas com “-” que não existam, serão processadas como uma opção ilegal.

```
tsora@DESKTOP-RRHS0DA:/mnt/c/Users/tiago/Documents/GitHub/SO$ ./netifstat.sh -h 10
./netifstat.sh: illegal option -- h
Uso de argumentos inválidos.
Menu de Uso e Execução do Programa.
-c [NETIF] : Seleção das interfaces de rede, [NETIF], a visualizar através de uma expressão regular.
-b          : Visualização das quantidades em bytes.
-k          : Visualização das quantidades em kilobytes.
-m          : Visualização das quantidades em megabytes.
-p [n]      : Número, [n], de interfaces de redes a visualizar.
-t          : Ordenação da tabela por TX (decrescente).
-r          : Ordenação da tabela por RX (decrescente)..
-T          : Ordenação da tabela por TRATE (decrescente).
-R          : Ordenação da tabela por RRATE (decrescente).
-v          : Ordenação reversa (crescente).
-l          : Loop de execução do programa a cada [s] segundos.
ALERTAS -> As opções -t,-r,-T,-R não podem ser utilizadas em simultâneo.
           O último argumento passado tem de o período de tempo desejado (segundos).
```

Img 40 – ./netifstat.sh -h 10

Para terminar a lista de todos os possíveis resultados inválidos, temos aqueles resultados que são originados pelo trabalho feito pela variável `$ctr`, variável global responsável por saber se o tudo o utilizador executou devidamente o código.

No primeiro caso, a execução é incorreta, já que 2 tanto é o argumento adicional de `-c` como também o último argumento do comando.

```
tsora@DESKTOP-RRHS0DA:/mnt/c/Users/tiago/Documents/GitHub/SO$ ./netifstat.sh -c 2
Uso de argumentos inválidos.
Menu de Uso e Execução do Programa.
-c [NETIF] : Seleção das interfaces de rede, [NETIF], a visualizar através de uma expressão regular.
-b          : Visualização das quantidades em bytes.
-k          : Visualização das quantidades em kilobytes.
-m          : Visualização das quantidades em megabytes.
-p [n]      : Número, [n], de interfaces de redes a visualizar.
-t          : Ordenação da tabela por TX (decrescente).
-r          : Ordenação da tabela por RX (decrescente)..
-T          : Ordenação da tabela por TRATE (decrescente).
-R          : Ordenação da tabela por RRATE (decrescente).
-v          : Ordenação reversa (crescente).
-l          : Loop de execução do programa a cada [s] segundos.
ALERTAS -> As opções -t,-r,-T,-R não podem ser utilizadas em simultâneo.
           O último argumento passado tem de o período de tempo desejado (segundos).
```

Img 41 – `./netifstat.sh -c 2`

No segundo caso, mesmo “p” não sendo considerado uma opção pelo programa (e, portanto, a única coisa que o programa realmente vai ler é o “10” como argumento final), o programa reconhece que o utilizador não utilizou o “-p” devidamente e, portanto, dá um aviso de argumentos inválidos.

```
tsora@DESKTOP-RRHS0DA:/mnt/c/Users/tiago/Documents/GitHub/SO$ ./netifstat.sh p 2 10
Uso de argumentos inválidos.
Menu de Uso e Execução do Programa.
-c [NETIF] : Seleção das interfaces de rede, [NETIF], a visualizar através de uma expressão regular.
-b          : Visualização das quantidades em bytes.
-k          : Visualização das quantidades em kilobytes.
-m          : Visualização das quantidades em megabytes.
-p [n]      : Número, [n], de interfaces de redes a visualizar.
-t          : Ordenação da tabela por TX (decrescente).
-r          : Ordenação da tabela por RX (decrescente)..
-T          : Ordenação da tabela por TRATE (decrescente).
-R          : Ordenação da tabela por RRATE (decrescente).
-v          : Ordenação reversa (crescente).
-l          : Loop de execução do programa a cada [s] segundos.
ALERTAS -> As opções -t,-r,-T,-R não podem ser utilizadas em simultâneo.
           O último argumento passado tem de o período de tempo desejado (segundos).
```

Img 42 – `./netifstat.sh p 2 10`

6 – Conclusão

O *script* desenvolvido, *netifstat.sh*, permite apresentar estatísticas sobre a quantidade de dados transmitidos e recebidos nas interfaces de rede selecionadas, e sobre as respetivas taxas de transferência. Permite ainda selecionar a informação a ser apresentada, através do uso de opções passadas no terminal aquando à execução.

Para além da possibilidade de criar este *script*, este trabalho também nos possibilitou momentos de aprendizagem. Aprendemos como inicializar e trabalhar com arrays associativos e com variáveis locais e globais, interiorizando o conceito de *key* e *value* de um array, o que nos permitiu resolver imensos problemas, principalmente, em relação ao tratamento e transporte dos valores retornados e calculados ao longo de todo o *script*. Ainda aprendemos como trabalhar melhor com diretórios, pastas e documentos, o que nos permitiu obter os dados dos valores que precisávamos para calcular a quantidade de dados transmitidos e recebidos nas interfaces de rede durante o intervalo de tempo pedido.

Foi durante o desenvolvimento do trabalho que fomos aprendendo e aperfeiçoando o nosso código, pelo que tivemos de rescrever o código imensas vezes e criar imensos repositórios no GitHub. Tivemos de ir aprendendo aos poucos como implementar o código necessário para responder às perguntas e exigências do guião, mas agora, com o código mais organizado e sintetizado, estamos contentes com o resultado obtido.

Atualmente, podemos dizer que conseguimos interpretar e responder devidamente ao guião apresentado pelo professor e podemos nos orgulhar de ter tido a oportunidade de ter feito este trabalho.

7 – Bibliografia

No decorrer da realização do trabalho e principalmente, na ajuda com as condições necessárias, para implementar todas as funcionalidades do programa, consultamos tanto os slides disponibilizados pelo Professor no *e-learning* na página da U.C. de Sistemas Operativos como também os sites que seguem na lista seguinte.

- ❖ <https://stackoverflow.com/>
- ❖ <https://man.cx/bash>
- ❖ <https://unix.stackexchange.com/questions/tagged/bash>
- ❖ <https://www.cyberciti.biz/faq/linux-list-network-interfaces-names-command/>
- ❖ <https://www.computerhope.com/unix/bash/getopts.htm>