Trabalho 1

Monitorização de interfaces de rede em bash

Universidade de Aveiro

Licenciatura em Engenharia Informática

Sistemas Operativos

6 dezembro 2021

Pedro Jorge, Nº Mecanográfico: 98418

Índice

Introdução

Como primeiro trabalho da cadeira Sistemas Operativos foi proposto desenvolver um script em bash, “netifstat.sh”, com o objetivo de apresentar estatísticas sobre a quantidade de dados transmitidos e recebidos nas interfaces de rede selecionadas, e sobre as respectivas taxas de transferências.

O script tem como parâmetro obrigatório o número de segundos que serão usados para calcular as taxas de transferência.

Existem vários parâmetros que podem ser passados na execução do código:

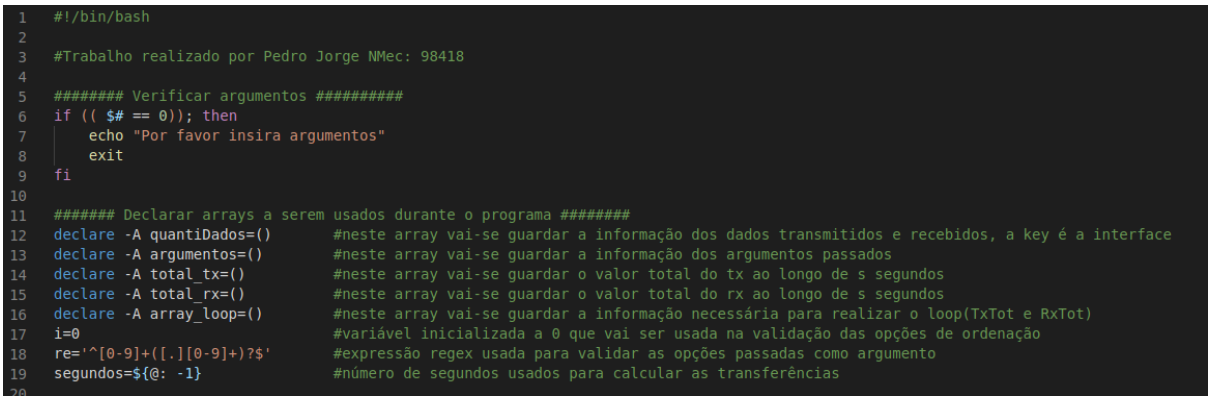
- **-c** : seleciona as interfaces a visualizar através de uma expressão regular;
- **-b** : visualizar dados em bytes;
- **-k** : visualizar dados em kilobytes;
- **-m** : visualizar dados em megabytes;
- **-p** : número de interfaces a visualizar;
- **-t** : ordenar, de forma decrescente, pelo Tx;
- **-r** : ordenar, de forma decrescente, pelo Rx;
- **-T** : ordenar, de forma decrescente, pelo TRATE;
- **-R** : ordenar, de forma decrescente, pelo RRATE;
- **-v** : ordenar em *reverse;*
- **-l** : loop, o script deve funcionar em loop, sem terminar, imprimindo a s segundos nova informação.

Abordagem Usada

De forma a resolver este problema, a abordagem usada foi dividir o problema em 3 partes: a primeira parte consiste em declarar todas as variáveis a usar, especificar os argumentos a usar e fazer a sua validação. A segunda parte consiste em criar uma função “principal” onde vai ser desenvolvido o código necessário para ir buscar os *bytes* transmitidos e recebidos do Tx e Rx e os restantes requisitos. E, por fim, a terceira parte está relacionada com o tratamento dos prints, formatação da tabela e especificar o que cada argumento faz.

Primeira Parte

Nesta primeira parte, e como foi referido anteriormente, o objetivo foi declarar as variáveis, especificar os argumentos e fazer a validação dos mesmos.



```
#!/bin/bash

#Trabalho realizado por Pedro Jorge NMec: 98418

######## Verificar argumentos ##########
if (( $# == 0)); then
    echo "Por favor insira argumentos"
    exit
fi

####### Declarar arrays a serem usados durante o programa ########
declare -A quantiDados=()       #neste array vai-se guardar a informação dos dados transmitidos e recebidos, a key é a interface
declare -A argumentos=()        #neste array vai-se guardar a informação dos argumentos passados
declare -A total_tx=()          #neste array vai-se guardar o valor total do tx ao longo de s segundos
declare -A total_rx=()          #neste array vai-se guardar o valor total do rx ao longo de s segundos
declare -A array_loop=()        #neste array vai-se guardar a informação necessária para realizar o loop(TxTot e RxTot)
i=0                             #variável inicializada a 0 que vai ser usada na validação das opções de ordenação
re='^[0-9]+([.][0-9]+)?$'       #expressão regex usada para validar as opções passadas como argumento
segundos=${@: -1}               #número de segundos usados para calcular as transferências
```

Fig 1 - Declaração de arrays e verificar se foi passado argumentos

Primeiramente, criou-se uma condição “if” que vai verificar se foram passados argumentos, caso contrário, imprime uma mensagem a dizer para o utilizador inserir argumentos e encerra o programa.

De seguida, criaram-se 5 arrays:

- **quantiDados:** neste array vai ser guardada toda a informação relativa aos dados transmitidos e recebidos do Tx e Rx e as taxas de transferência TRate e RRate. A key vai ser o nome da interface;

- **argumentos:** neste array vai-se guardar todos os argumentos passados pelo utilizador;

- **total_tx:** este array é usado na opção “loop”, vai guardar e incrementar o delta Tx(diferença entre o Tx calculado inicialmente e o Tx calculado após o *sleep*) ao longo de **s** segundos;

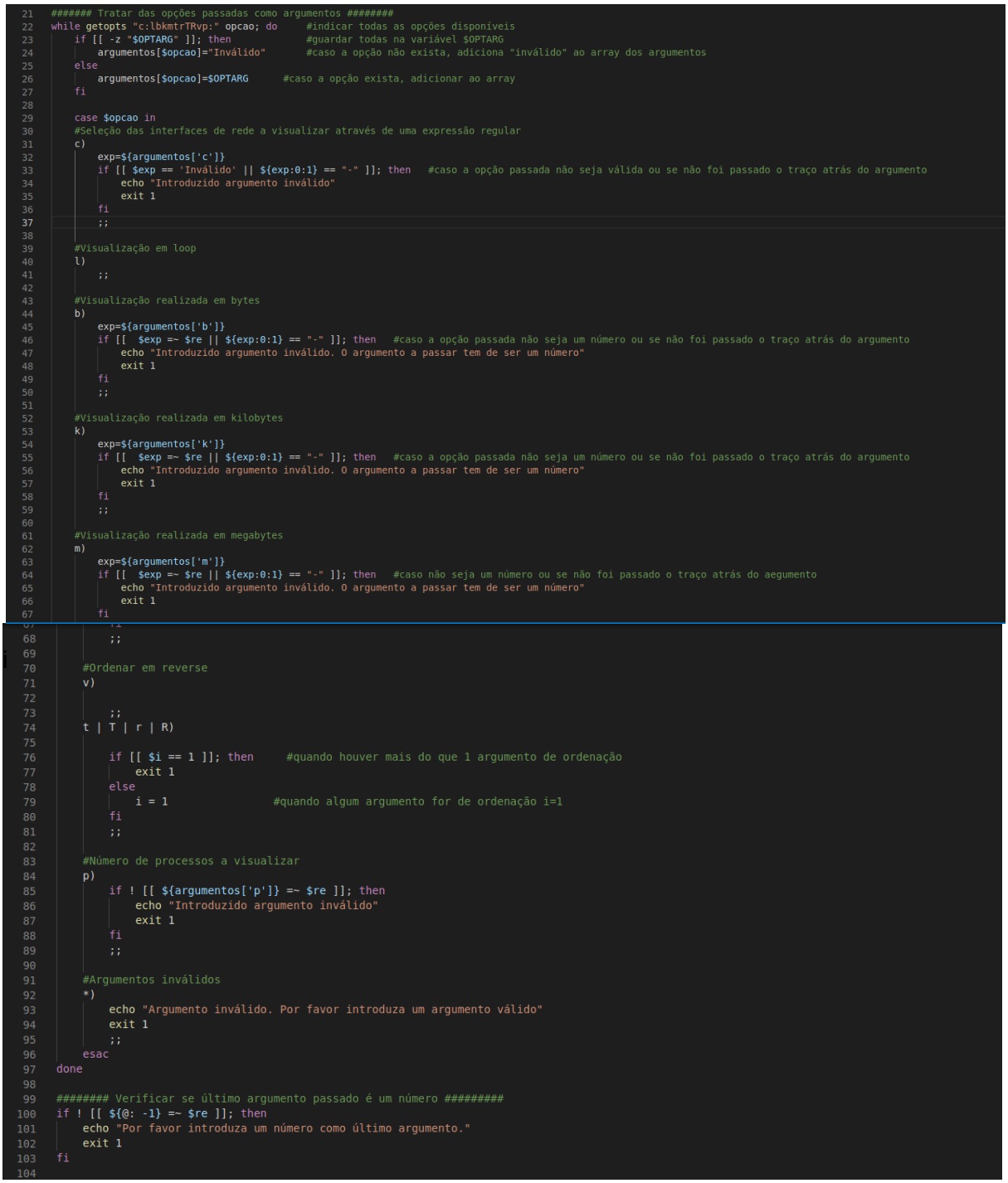
- **total_rx:** array que vai desempenhar a mesma função que o anterior, mas para o Rx;

- **array_loop:** este array é bastante similar ao array “quantiDados”, com a única diferença sendo que vai ser usado exclusivamente na opção “loop” e vai guardar também a informação relativa ao “total_tx” e “total_rx”.

Inicializou-se também a variável **“i”** a zero, esta variável vai ser usada para verificar se as opções de ordenação vão ser usadas ao mesmo tempo, o que não é permitido.

Criou-se a variável **“re”** que vai validar se o valor passado à frente do argumento é um número.

Por fim, criou-se a variável “**segundos”** que vai corresponder ao número de segundos que o utilizador quer usar como tempo para calcular as taxas de transferência. Vai ser usada como tempo de *sleep.*



```
###### Tratar das opções passadas como argumentos ########
while getopts "c:lbkmtrTRvp:" opcao; do     #indicar todas as opções disponiveis
    if [[ -z "$OPTARG" ]]; then             #guardar todas na variável $OPTARG
        argumentos[$opcao]="Inválido"       #caso a opção não exista, adiciona "inválido" ao array dos argumentos
    else
        argumentos[$opcao]=$OPTARG      #caso a opção exista, adicionar ao array
    fi

    case $opcao in
    #Seleção das interfaces de rede a visualizar através de uma expressão regular
    c)
        exp=${argumentos['c']}
        if [[ $exp == 'Inválido' || ${exp:0:1} == "-" ]]; then   #caso a opção passada não seja válida ou se não foi passado o traço atrás do argumento
            echo "Introduzido argumento inválido"
            exit 1
        fi
        ;;

    #Visualização em loop
    l)
        ;;

    #Visualização realizada em bytes
    b)
        exp=${argumentos['b']}
        if [[  $exp =~ $re || ${exp:0:1} == "-" ]]; then   #caso a opção passada não seja um número ou se não foi passado o traço atrás do argumento
            echo "Introduzido argumento inválido. O argumento a passar tem de ser um número"
            exit 1
        fi
        ;;

    #Visualização realizada em kilobytes
    k)
        exp=${argumentos['k']}
        if [[  $exp =~ $re || ${exp:0:1} == "-" ]]; then   #caso a opção passada não seja um número ou se não foi passado o traço atrás do argumento
            echo "Introduzido argumento inválido. O argumento a passar tem de ser um número"
            exit 1
        fi
        ;;

    #Visualização realizada em megabytes
    m)
        exp=${argumentos['m']}
        if [[  $exp =~ $re || ${exp:0:1} == "-" ]]; then   #caso não seja um número ou se não foi passado o traço atrás do aegumento
            echo "Introduzido argumento inválido. O argumento a passar tem de ser um número"
            exit 1
        fi
        ;;

    #Ordenar em reverse
    v)

        ;;
    t | T | r | R)

        if [[ $i == 1 ]]; then     #quando houver mais do que 1 argumento de ordenação
            exit 1
        else
            i = 1               #quando algum argumento for de ordenação i=1
        fi
        ;;

    #Número de processos a visualizar
    p)
        if ! [[ ${argumentos['p']} =~ $re ]]; then
            echo "Introduzido argumento inválido"
            exit 1
        fi
        ;;

    #Argumentos inválidos
    *)
        echo "Argumento inválido. Por favor introduza um argumento válido"
        exit 1
        ;;
    esac
done

######## Verificar se último argumento passado é um número #########
if ! [[ ${@: -1} =~ $re ]]; then
    echo "Por favor introduza um número como último argumento."
    exit 1
fi
```

Fig 2 - Validar opções passadas como argumentos

Para finalizar a primeira parte da implementação, foi dada ao script a capacidade de interpretar diferentes argumentos, bem como a possibilidade de aceitar argumentos desses argumentos. Para isso utilizou-se a função **getopts** (onde se especificou todos as opções

possíveis de serem usados). De seguida, tratou-se da validação de cada uma das opções. No caso do **-c, -b, -k e -m** criou-se uma condição que verifica se a opção é inválida (não pertence às opções já predefinidos) e se o primeiro caracter passado fosse um traço (isto não pode acontecer porque pode-e correr o risco de uma opção assumir como valor outro uma opção passada como argumento). No caso do **-l,** não se fez nenhuma validação visto que esta opção vai ser apenas usada para fazer *loop* e não leva nada à frente. Para a opção **-v** não se fez nenhuma validação específica visto que esta opção faz apenas o *reverse* das operações **-t**, **-T**, **-r** e **-R.** Para estas opções, a validação que se fez foi impedir que sejam usados em simultâneo, que funciona da seguinte forma:

- Após a variável "i" ter sido inicializada com valor zero, e, imaginemos, serem passados os argumentos -t -T, o *getopts* vai verificar se o valor de i é igual a um, caso não seja vai fazer i = 1;
- De seguida, o *getopts* vai verificar o valor de i na opção -T e, caso seja 1, o programa encerra.

Por fim, quanto à opção **-p,** visto que esta só pode aceitar números como argumentos, criou-se uma condição que verifica se o argumento passado não faz parte da expressão "re" definida anteriormente. Em caso negativo, é impressa uma mensagem de erro e o programa encerra.

Por fim, fez-se a verificação se o último argumento passado é um número. Visto que este projeto tem como parâmetro obrigatório o tempo usado para calcular as taxas de transferência, caso não seja passado um número como último argumento é impressa uma mensagem de erro e o programa encerra.

Segunda Parte

No que toca à segunda parte, esta foca-se na obtenção dos valores de Tx e Rx de cada interface, calcular as taxas de transferência de cada interface e converter todos os valores para kilobytes ou megabytes, consoante o utilizador desejar. Tudo isto foi implementado na mesma função

```
105  ######## Função para tratar a quantidade de dados recebidos e envidados nas interfaces de rede
106  function principal() {
107      for iface in $(ifconfig | cut -d ' ' -f1| tr ':' '\n' | awk NF)
108      do
109          tx1=$(cat /sys/class/net/$iface/statistics/tx_bytes)
110          rx1=$(cat /sys/class/net/$iface/statistics/rx_bytes)
111          if [[ $tx1 == 0 && $rx1 == 0 ]]; then
112              continue
113          else
114              TX1[$iface]=$(printf "%12d\n" "$tx1")
115              RX1[$iface]=$(printf "%12d\n" "$rx1")
116          fi
117      done
118
119      sleep $segundos  #segundos a usar para calcular a transferência de dados, primeiro parâmetro a ser passado
120
121      for iface in $(ifconfig | cut -d ' ' -f1| tr ':' '\n' | awk NF)
122      do
123          tx2=$(cat /sys/class/net/$iface/statistics/tx_bytes)
124          rx2=$(cat /sys/class/net/$iface/statistics/rx_bytes)
125          if [[ $tx2 == 0 && $rx2 == 0 ]]; then                #ao implementar esta opção todas as interfaces vão aparecer na tabela mesmo que contenham o valor 0
126              diftx=0                                      #de forma a aparecerem as interfaces basta descomentar estas linhas todas
127              difrx=0
128              tRate=0
129              rRate=0
130              quantiDados[$iface]=$(printf "%-15s %-10s %10s %10s %10s\n" "$iface" "$diftx" "$difrx" "$tRate" "$rRate")
131              continue
132          else
133              diftx=$(($tx2-${TX1[$iface]}))
134              difrx=$(($rx2-${RX1[$iface]}))
135
136              #### Mostrar valores em Bytes ###
137              if [[ -v argumentos[b] ]]; then
138                  tRate=$(echo "scale=1; $diftx/$1" | bc -l)
139                  rRate=$(echo "scale=1; $difrx/$1" | bc -l)
140                  quantiDados[$iface]=$(printf "%-15s %-10s %10s %10s %10s\n" "$iface" "$diftx" "$difrx" "$tRate" "$rRate")
141
142              #### Mostrar valores em Kilobytes ####
143              elif [[ -v argumentos[k] ]]; then
144                  kilobyte=1024
145                  diftxKilo=$(echo "scale=3; $diftx/$kilobyte" | bc -l)
146                  difrxKilo=$(echo "scale=3; $diftx/$kilobyte" | bc -l)
147                  tRate=$(echo "scale=3; $diftxKilo/$1" | bc -l)
148                  rRate=$(echo "scale=3; $difrxKilo/$1" | bc -l)
149                  tRate=${tRate/#./0.}     #adicionar 0 atrás, questão de estética
150                  rRate=${rRate/#./0.}     #adicionar 0 atrás, questão de estética
151                  quantiDados[$iface]=$(printf "%-15s %-10s %10s %10s %10s\n" "$iface" "$diftxKilo" "$difrxKilo" "$tRate" "$rRate")
152
153              ####Mostrar valores em Megabytes ######
154              elif [[ -v argumentos[m] ]]; then
155                  megabyte=1048576            #multiplicar 1024*1024 para obter os megabytes
156                  diftxMega=$(echo "scale=3; $diftx/$megabyte" | bc -l)          #3 casas decimais de forma se possível observar o resultado
157                  difrxMega=$(echo "scale=3; $diftx/$megabyte" | bc -l)          #3 casas decimais de forma se possível observar o resultado
158                  tRate=$(echo "scale=3; $diftxMega/$1" | bc -l)                 #3 casas decimais de forma se possível observar o resultado
159                  rRate=$(echo "scale=3; $difrxMega/$1" | bc -l)                 #3 casas decimais de forma se possível observar o resultado
160                  tRate=${tRate/#./0.}     #adicionar 0 atrás, questão de estética
161                  rRate=${rRate/#./0.}     #adicionar 0 atrás, questão de estética
162                  quantiDados[$iface]=$(printf "%-15s %-10s %10s %10s %10s\n" "$iface" "$diftxMega" "$difrxMega" "$tRate" "$rRate")
163
164              #### Caso não seja passado nenhum argumento específico, mostrar valores em bytes####3
165              else
166                  tRate=$(echo "scale=1; $diftx/$1" | bc -l)
167                  rRate=$(echo "scale=1; $difrx/$1" | bc -l)
168                  quantiDados[$iface]=$(printf "%-15s %-10s %10s %10s %10s\n" "$iface" "$diftx" "$difrx" "$tRate" "$rRate")
169              fi
170          fi
171      done
```

Fig 3 - Primeira parte da função principal, obtenção Tx e Rx, cálculo taxas, etc.

Primeiramente, criou-se 2 ciclos “for” onde se calculou os dados (em bytes) do Tx e do Rx, sendo que a separá-los está o comando sleep. Isto permite que o código seja

eficiente visto que consegue-se fazer sleep uma vez durante toda a execução do código.
Ambos os ciclos “for” percorrem todas as interfaces de rede do sistema.

```
105  ######## Função para tratar a quantidade de dados recebidos e envidados nas interfaces de rede
106  function principal() {
107      for iface in $(ifconfig | cut -d ' ' -f1| tr ':' '\n' | awk NF)
108      do
109          tx1=$(cat /sys/class/net/$iface/statistics/tx_bytes)
110          rx1=$(cat /sys/class/net/$iface/statistics/rx_bytes)
111          if [[ $tx1 == 0 && $rx1 == 0 ]]; then
112              continue
113          else
114              TX1[$iface]=$(printf "%12d\n" "$tx1")
115              RX1[$iface]=$(printf "%12d\n" "$rx1")
116          fi
117      done
118
119      sleep $segundos  #segundos a usar para calcular a transferência de dados, primeiro parâmetro a ser passado
120
```

Fig. 4 - Primeiro ciclo “for”

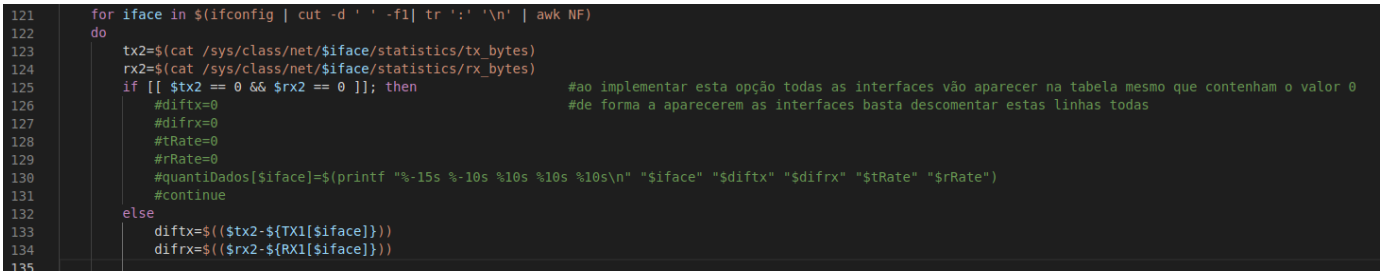
No primeiro ciclo *“for”* (figura 4), apenas se calculou os primeiros/iniciais valores de Tx e de Rx através do comando ***“cat”*** que vai percorrer o diretório "sys/class/net/(interface)/statistics/(r)tx bytes" e retornar o valor em bytes do Tx e Rx e guardar nas respetivas variáveis.

Implementou-se também uma condição que vai verificar se os valores de Tx e Rx iniciais são zero. Caso isto se verifique, o comando *“continue”* faz com que o código avance para o próximo processo do ciclo *“for”.* Caso contrário, os valores de Tx e Rx são adicionados aos arrays associativos TX1[(interface)] e RX1[(interface)] onde a *key* vai ser o nome das interfaces. Caso isto não seja aplicado corre-se o risco de cometer erros iguais ao da figura abaixo aquando da execução do código.

```
./testes.sh: linha 24: br-0701a2b3c077: valor muito grande para a base (símbolo
de erro é "0701a2b3c077")
./testes.sh: linha 26: br-0701a2b3c077: valor muito grande para a base (símbolo
de erro é "0701a2b3c077")
```

Fig. 5 - Erro caso não se implemente as verificações

Por fim, e depois de acabar o ciclo *“for”* , é executado o comando sleep onde lhe é passado o argumento “*$**segundos**”*, correspondente ao tempo usado para calcular as taxas de transferência.



```
for iface in $(ifconfig | cut -d ' ' -f1| tr ':' '\n' | awk NF)
do
    tx2=$(cat /sys/class/net/$iface/statistics/tx_bytes)
    rx2=$(cat /sys/class/net/$iface/statistics/rx_bytes)
    if [[ $tx2 == 0 && $rx2 == 0 ]]; then                #ao implementar esta opção todas as interfaces vão aparecer na tabela mesmo que contenham o valor 0
        #diftx=0                                         #de forma a aparecerem as interfaces basta descomentar estas linhas todas
        #difrx=0
        #tRate=0
        #rRate=0
        #quantiDados[$iface]=$(printf "%-15s %-10s %10s %10s %10s\n" "$iface" "$diftx" "$difrx" "$tRate" "$rRate")
        #continue
    else
        diftx=$(($tx2-${TX1[$iface]}))
        difrx=$(($rx2-${RX1[$iface]}))
```

Fig. 6 - Segundo ciclo “for”

No que toca ao segundo ciclo *“for”*, o processo é semelhante ao primeiro ciclo *“for”.* Começa-se por voltar a calcular os valores de Tx e Rx e guardar nas variáveis ***“tx2”*** e ***“rx2”*** respetivamente.
De seguida volta-se a implementar uma condição que vai verificar se os valores são iguais a zero. Neste caso é possível visualizar todos os processos disponíveis (mesmo os que têm os valores de Tx e Rx a zero) através do descomentar das linhas de código dentro da condição. Caso esta condição não se verifique, vai ser calculado a diferença entre o Tx inicial e o Tx calculado agora e a diferença entre o Rx inicial e o Rx acabado de calcular e guarda-se nas variáveis ***“diftx”*** e ***“difrx”.***

```
136             #### Mostrar valores em Bytes ###
137             if [[ -v argumentos[b] ]]; then
138                 tRate=$(echo "scale=1; $diftx/$1" | bc -l)
139                 rRate=$(echo "scale=1; $difrx/$1" | bc -l)
140                 quantiDados[$iface]=$(printf "%-15s %-10s %10s %10s %10s\n" "$iface" "$diftx" "$difrx" "$tRate" "$rRate")
141
142             #### Mostrar valores em Kilobytes ####
143             elif [[ -v argumentos[k] ]]; then
144                 kilobyte=1024
145                 diftxKilo=$(echo "scale=3; $diftx/$kilobyte" | bc -l)
146                 difrxKilo=$(echo "scale=3; $diftx/$kilobyte" | bc -l)
147                 tRate=$(echo "scale=3; $diftxKilo/$1" | bc -l)
148                 rRate=$(echo "scale=3; $difrxKilo/$1" | bc -l)
149                 tRate=${tRate/#./0.}     #adicionar 0 atrás, questão de estética
150                 rRate=${rRate/#./0.}     #adicionar 0 atrás, questão de estética
151                 quantiDados[$iface]=$(printf "%-15s %-10s %10s %10s %10s\n" "$iface" "$diftxKilo" "$difrxKilo" "$tRate" "$rRate")
152
153             ####Mostrar valores em Megabytes ######
154             elif [[ -v argumentos[m] ]]; then
155                 megabyte=1048576            #multiplicar 1024*1024 para obter os megabytes
156                 diftxMega=$(echo "scale=3; $diftx/$megabyte" | bc -l)           #3 casas decimais de forma se possível observar o resultado
157                 difrxMega=$(echo "scale=3; $diftx/$megabyte" | bc -l)           #3 casas decimais de forma se possível observar o resultado
158                 tRate=$(echo "scale=3; $diftxMega/$1" | bc -l)                  #3 casas decimais de forma se possível observar o resultado
159                 rRate=$(echo "scale=3; $difrxMega/$1" | bc -l)                  #3 casas decimais de forma se possível observar o resultado
160                 tRate=${tRate/#./0.}     #adicionar 0 atrás, questão de estética
161                 rRate=${rRate/#./0.}     #adicionar 0 atrás, questão de estética
162                 quantiDados[$iface]=$(printf "%-15s %-10s %10s %10s %10s\n" "$iface" "$diftxMega" "$difrxMega" "$tRate" "$rRate")
163
164             #### Caso não seja passado nenhum argumento específico, mostrar valores em bytes####3
165             else
166                 tRate=$(echo "scale=1; $diftx/$1" | bc -l)
167                 rRate=$(echo "scale=1; $difrx/$1" | bc -l)
168                 quantiDados[$iface]=$(printf "%-15s %-10s %10s %10s %10s\n" "$iface" "$diftx" "$difrx" "$tRate" "$rRate")
169             fi
170         fi
```

Fig. 7 - Última parte do segundo ciclo for

Por fim, e ainda dentro do ciclo *"for"*, começou-se por definir o que cada opção vai fazer:

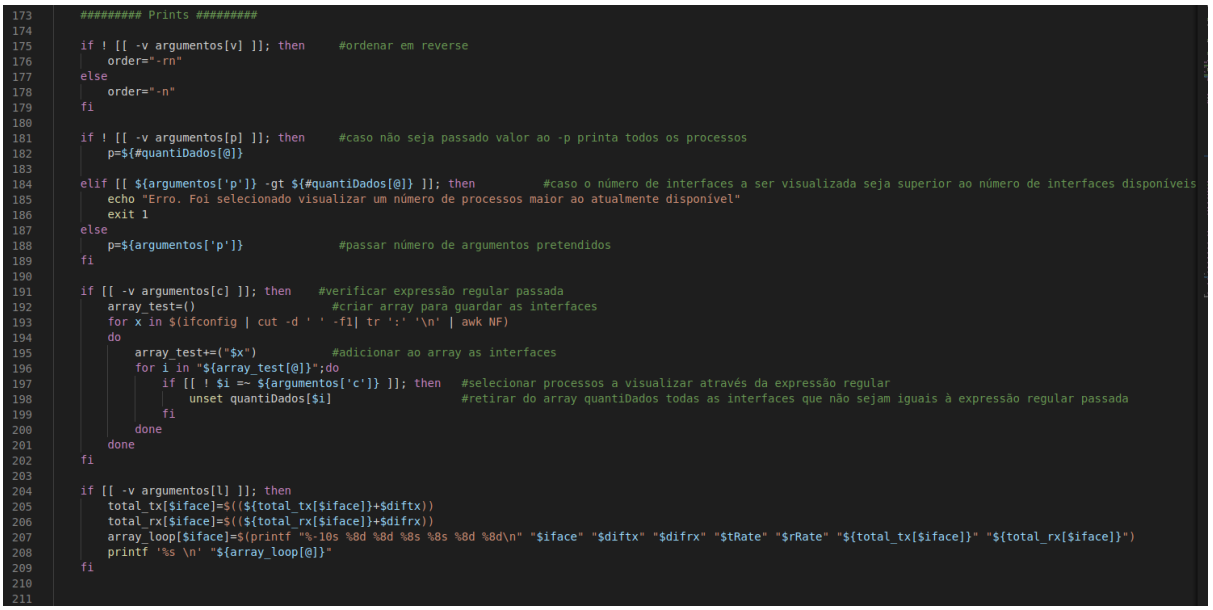
- Opção **-b:** Nesta parte do código fez-se o procedimento normal de maneira a calcular as taxas de transferência do Tx e do Rx. Para ambos dividiu-se os valores guardados nas variáveis ***"diftx"*** e ***"difrx"*** pelo valor armazenado na variável ***"segundos".***
  Aqui a impressão dos dados é feita em *bytes;*

- Opção **-k:** a implementação desta opção é bastante semelhante à opção anterior sendo a única diferença o uso de *kilobytes* em vez de *bytes.* Para tal bastou-se dividir as variáveis ***"diftx"*** e ***"difrx"*** por 1024, ajustar o nível de casas decimais adequado e fazer o resto do procedimento. Implementou-se também a opção de adicionar 0 nos casos do número ser decimal e menor que 1;

- Opção **-m:** novamente, a implementação desta opção é quase igual às anteriores, sendo a única a diferença o uso de *megabytes.* Para tal, dividiu-se as variáveis ***“diftx”*** e ***“difrx”*** por 1048576 (1024*1024), ajustar o nível de casas decimais adequado e fazer o resto do procedimento.

Por fim, caso nenhuma destas opções seja usada como argumento, os dados são impressos em bytes.

Terceira Parte



```
######### Prints #########

if ! [[ -v argumentos[v] ]]; then     #ordenar em reverse
    order="-rn"
else
    order="-n"
fi

if ! [[ -v argumentos[p] ]]; then     #caso não seja passado valor ao -p printa todos os processos
    p=${#quantiDados[@]}

elif [[ ${argumentos['p']} -gt ${#quantiDados[@]} ]]; then          #caso o número de interfaces a ser visualizada seja superior ao número de interfaces disponíveis
    echo "Erro. Foi selecionado visualizar um número de processos maior ao atualmente disponível"
    exit 1
else
    p=${argumentos['p']}              #passar número de argumentos pretendidos
fi

if [[ -v argumentos[c] ]]; then    #verificar expressão regular passada
    array_test=()                   #criar array para guardar as interfaces
    for x in $(ifconfig | cut -d ' ' -f1| tr ':' '\n' | awk NF)
    do
        array_test+=("$x")          #adicionar ao array as interfaces
        for i in "${array_test[@]}";do
            if [[ ! $i =~ ${argumentos['c']} ]]; then   #selecionar processos a visualizar através da expressão regular
                unset quantiDados[$i]                   #retirar do array quantiDados todas as interfaces que não sejam iguais à expressão regular passada
            fi
        done
    done
fi

if [[ -v argumentos[l] ]]; then
    total_tx[$iface]=$((${total_tx[$iface]}+$diftx))
    total_rx[$iface]=$((${total_rx[$iface]}+$difrx))
    array_loop[$iface]=$(printf "%-10s %8d %8d %8s %8s %8d %8d\n" "$iface" "$diftx" "$difrx" "$tRate" "$rRate" "${total_tx[$iface]}" "${total_rx[$iface]}")
    printf '%s \n' "${array_loop[@]}"
fi
```

Fig. 8 - Terceira e última parte (prints). Opções -v, -p, -c e -l

Para a terceira e última parte, esta foi dedicada ao tratamento dos prints, formatação da tabela e especificar o que cada argumento faz.

No que toca à especificação do que cada argumento faz, esta foi a implementação adotada:

- Opção **-v:** Criou-se uma condição *"if"*, onde se vai verificar se a opção -v foi usada. Caso tenha sido usada, a variável *"order"* passa a "***rn",*** que ordenar por ordem decrescente. Caso contrário a variável *"order"* passa a "***n"*** e aí as informações são impressas por ordem crescente;
- Opção **-p:** Criou-se uma outra condição *"if"*, onde se vai verificar se foram passados argumentos à opção **-p.** Se não foram passados argumentos, todos os processos vão ser impressos. Caso algum argumento

seja passado, vão ser impressos o número de processos passados como argumento. Foi também implementada uma condição que verifica se o número de processos a visualizar é maior do que o número de processos atualmente guardados no array ***"quantiDados[@]".***

- Opção **-c:** De maneira a implementar esta opção criou-se um array, ***"arraytest",*** que vai guardar todas as interfaces. De seguida inicia-se um ciclo *"for"* que vai buscar todas as interfaces de rede e adicionar ao array. Por fim, inicia-se outro ciclo *"for"* que vai percorrer o array e caso a expressão regular não seja igual a nenhuma interface que esteja no array vamos remover essa interface do array através do comando ***unset.***
- Opção **-I:** Quanto ao loop, adicionou-se aos arrays criados inicialmente (***total_tx*** e ***total_rx***) o valor do ***diftx*** e ***difrx*** respetivamente, sendo a key destes arrays a interface. Isto vai permitir que a cada s segundos o valor dos arrays esteja sempre a ser incrementado pelo valor do ***diftx*** e ***difrx.*** Por fim a informação é toda impressa através do array associativo ***"array_loop[(interface)]".***

```
    if [[ -v argumentos[t] ]]; then       #ordenar pela pelo Tx
        printf '%s \n' "${quantiDados[@]}" | sort $order -k2 | head -n $p

    elif [[ -v argumentos[r] ]]; then       #ordenar pela pelo Rx
        printf '%s \n' "${quantiDados[@]}" | sort $order -k3 | head -n $p

    elif [[ -v argumentos[T] ]]; then       #ordenar pela pelo TRate
        printf '%s \n' "${quantiDados[@]}" | sort $order -k4 | head -n $p

    elif [[ -v argumentos[R] ]]; then       #ordenar pela pelo RRate
        printf '%s \n' "${quantiDados[@]}" | sort $order -k5 | head -n $p

    else                                    #ordenar pela ordem alfabética dos procesoss
        order="-n"
        printf '%s \n' "${quantiDados[@]}" | sort $order -k1 | head -n $p
    fi
}

function escolha(){
    if [[ ! -v argumentos[l] ]]; then
        printf "%-15s %-10s %10s %10s %10s\n" "NETIF" "TX" "RX" "TRATE" "RRATE"
        principal ${@: -1}
    else
        printf "%10s %8s %8s %8s %8s %8s %8s %8s\n" "NETIF" "TX" "RX" "TRATE" "RRATE" "TXTOT" "RXTOT"
        while true
        do
            principal ${@: -1}
        done
    fi
}
escolha ${@: -1}
```

Figura 8 - Última parte da função principal e função escolha

Quanto à última fase da especificação de cada argumento, todos têm uma implementação muito semelhante. Para cada um é criado uma condição “if” (ou “elif”) que verifica se o argumento foi passado. O operador “-v” vai verificar se cada opção é uma das keys guardadas no array “***argumentos”***. Caso seja passado acontece o seguinte no caso da:

- Opção **-t:** Ordenar pelo Tx. Para isso vai-se dar print a todo o conteúdo que o array ***“quantiDados”*** contém, seguido do comando *sort* que vai ordenar a informação pela opção pretendida, sendo a opção **“k”** que especifica o local onde se pretende ordenar(por exemplo, “-**k1**” é referente à primeira posição) e, por fim, o comando *head* que permite imprimir quantos processos o utilizador quiser. Caso a opção **-p** não tenha sido utilizada este comando vai permitir imprimir todos os processos.

- Opção **-r:** Ordenar pelo Rx. O resto do procedimento é semelhante ao anterior, exceto a posição definida no comando *sort*;
- Opção **-T:** Ordenar pelo TRate. O resto do procedimento é semelhante ao anterior, exceto a posição definida no comando *sort;*
- Opção **-R:** Ordenar pelo RRate. O resto do procedimento é semelhante ao anterior, exceto a posição definida no comando *sort;*

Caso não sejam usadas nenhuma destas opções, a variável ***"order"*** assume o valor "-n", que vai permitir ordenar o nome das interfaces pela ordem alfabética e aplica-se o mesmo procedimento usado nos casos anteriores.

Por fim, foi ainda criada a função **"Escolha"** que vai ajudar na formatação das tabelas. Nesta função é criada uma condição ***"if"*** que vai verificar se o argumento ***"l"*** não foi passado. Caso não tenha sido passado, imprime-se o cabeçalho da tabela e invoca-se a função principal, que vai permitir ir buscar todos os valores e imprimir esses valores. Caso contrário, imprime-se a tabela que deve aparecer ao executar a opção "l" (em relação à outra tabela adicionou-se "TxTot" e "RxTot" a esta), inicia-se um ciclo ***"while true"*** de modo a executar o loop, e dentro desse ciclo volta-se a invocar a função principal.

Resultados/Testes Realizados

De maneira a validar a solução e os resultados obtidos, foram feitos vários testes: uns para corresponder ao pedido no enunciado, e outros para testar a prevenção de erros.

Passando aos resultados obtidos:

./netifstat.sh 10 - executar passando só o número de segundos como argumento;

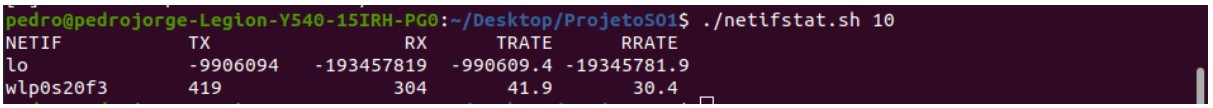


Fig. 10 - ./netifstat.sh 10

./netifstat.sh -b 10 - executar passando só o número de segundos como argumento e imprimir resultado em bytes;

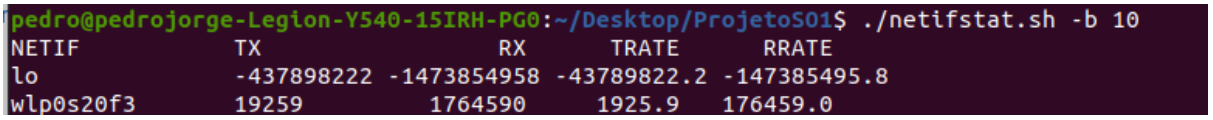


Fig. 11 - ./netifstat.sh -b 10

./netifstat.sh -k 10 - executar passando só o número de segundos como argumento e imprimir resultado em kilobytes;

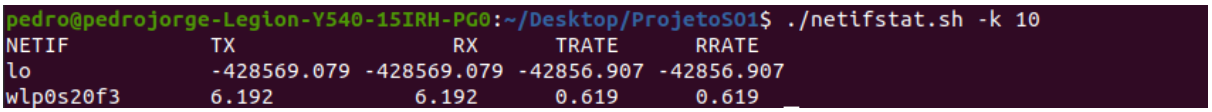


Fig. 12 - ./netifstat.sh -k 10

./netifstat.sh -m 10 - executar passando só o número de segundos como argumento e imprimir resultado em megabytes;

```
pedro@pedrojorge-Legion-Y540-15IRH-PG0:~/Desktop/ProjetoSO1$ ./netifstat.sh -m 10
NETIF           TX                RX       TRATE       RRATE
lo              -419.254     -419.254    -41.925     -41.925
wlp0s20f3       .009              .009          0           0
```

Fig. 13 - ./netifstat.sh -m 10

./netifstat.sh -t 10 - ordenar pelo Tx;

```
pedro@pedrojorge-Legion-Y540-15IRH-PG0:~/Desktop/ProjetoSO1$ ./netifstat.sh -t 10
./netifstat.sh: linha 79: i: comando não encontrado
NETIF           TX                RX       TRATE       RRATE
wlp0s20f3       18108             5720     1810.8       572.0
lo              -440909285 -1479161703 -44090928.5 -147916170.3
```

Fig. 14 - ./netifstat.sh -t 10

./netifstat.sh -T 10 - ordenar pelo TRate;

```
pedro@pedrojorge-Legion-Y540-15IRH-PG0:~/Desktop/ProjetoSO1$ ./netifstat.sh -T 10
./netifstat.sh: linha 79: i: comando não encontrado
NETIF           TX                RX       TRATE       RRATE
wlp0s20f3       1215              212      121.5        21.2
lo              -441445840 -1479309737 -44144584.0 -147930973.7
```

Fig. 15 - ./netifstat.sh -T 10

./netifstat.sh -r 10 - ordenar pelo Rx;

```
pedro@pedrojorge-Legion-Y540-15IRH-PG0:~/Desktop/ProjetoSO1$ ./netifstat.sh -r 10
./netifstat.sh: linha 79: i: comando não encontrado
NETIF           TX                RX       TRATE       RRATE
wlp0s20f3       6775              3868     677.5        386.8
lo              -441877343 -1479403877 -44187734.3 -147940387.7
```

Fig. 16 - ./netifstat.sh -r 10

./netifstat.sh -R 10 - ordenar pelo RRate;

```
pedro@pedrojorge-Legion-Y540-15IRH-PG0:~/Desktop/ProjetoSO1$ ./netifstat.sh -R 10
./netifstat.sh: linha 79: i: comando não encontrado
NETIF           TX                 RX       TRATE       RRATE
wlp0s20f3       307                526       30.7        52.6
lo              -442563914 -1488729063 -44256391.4 -148872906.3
```

Fig. 17 - ./netifstat.sh -R 10

./netifstat.sh -c "w.*" 10 - procurar por interface por nome ou número;

```
pedro@pedrojorge-Legion-Y540-15IRH-PG0:~/Desktop/ProjetoSO1$ ./netifstat.sh -c "w.*" -t  10
./netifstat.sh: linha 79: i: comando não encontrado
NETIF           TX                 RX       TRATE       RRATE
wlp0s20f3       21305              4349     2130.5      434.9
```

Fig. 18 - ./netifstat.sh -c "l.*" 10

./netifstat.sh -c 2 10 - procurar por interface por nome ou número;

```
pedro@pedrojorge-Legion-Y540-15IRH-PG0:~/Desktop/ProjetoSO1$ ./netifstat.sh -c 2 10
NETIF           TX                 RX       TRATE       RRATE
wlp0s20f3       35056              10468    3505.6      1046.8
```

Fig. 19 - ./netifstat.sh -c 2 10

./netifstat.sh -v -t 10 - ordenar de forma reversa o TxRate;

```
pedro@pedrojorge-Legion-Y540-15IRH-PG0:~/Desktop/ProjetoSO1$ ./netifstat.sh -v -t 10
./netifstat.sh: linha 79: i: comando não encontrado
NETIF           TX                 RX       TRATE       RRATE
lo              -455655485 -1545428895 -45565548.5 -154542889.5
wlp0s20f3       94492              8667     9449.2      866.7
```

./netifstat.sh -l 10 - executar loop sem terminar

Nota: de todos os comandos desenvolvidos este foi o que não ficou totalmente funcional, a interface “wlp0s20f3” aparece 2 vezes num grupo de 3 interfaces de cada vez, contudo, a primeira instância da interface vai sempre somando os valores ao array do TxTot e RxTot, o que comprova que essa parte está funcional;

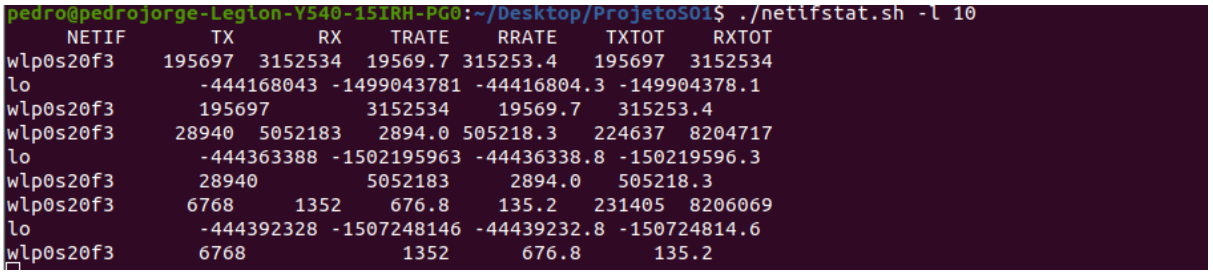


Fig. 20 - ./netifstat.sh -l 10

./netifstat.sh -p 1 10 - selecionar número de processos a visualizar, neste caso 1;

Fig. 21 - ./netifstat.sh -p 1 10

./netifstat.sh -c “l.*” -t 10 - combinar 2 argumentos: procurar interfaces com “l” no nome e ordenar pelo Tx

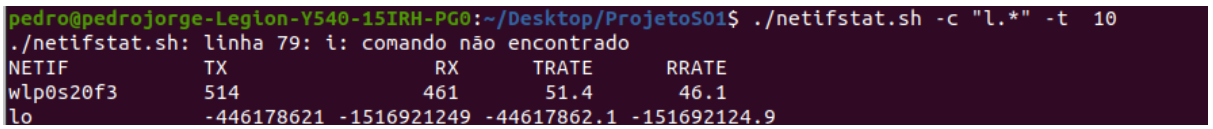


Fig. 21 - ./netifstat.sh -c “l.*” -t 10

./netifstat.sh -k -t 10 - ordenar valores em Kilobytes, pelo Tx

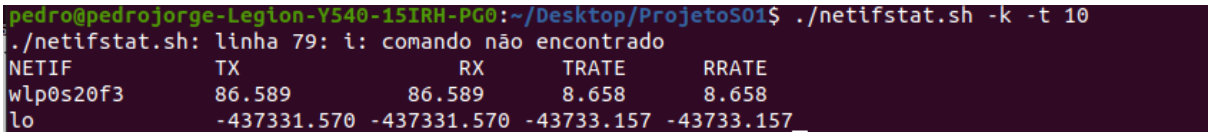


Fig. 22 - ./netifstat.sh -k -t 10

./netifstat.sh -k -c l -r 10 - usar 3 argumentos, ver todos os processos que contêm “l” no nome, em kilobytes e ordenado pelo Rx.

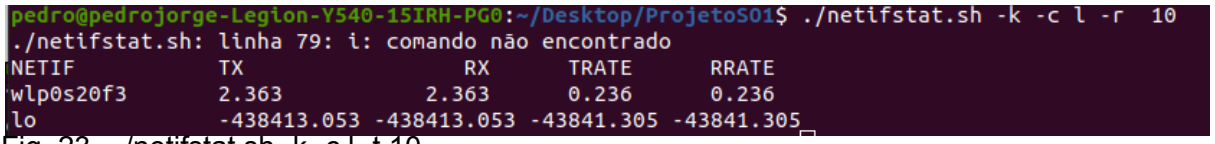


Fig. 23 - ./netifstat.sh -k -c l -t 10

./netifstat.sh -p 1 -k -c l -r 10 - usar 4 argumentos, ver apenas 1 processo que contém “l” no nome, em kilobytes e que tem Rx maior.

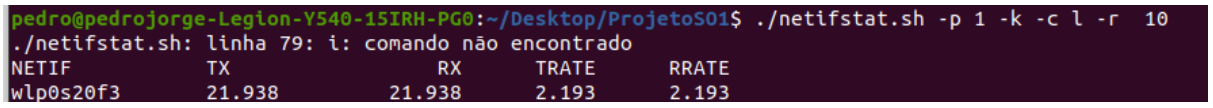


Fig. 23 - ./netifstat.sh -k -c l -t 10

Testando resultados inválidos:

./netifstat.sh - correr o programa sem passar argumentos

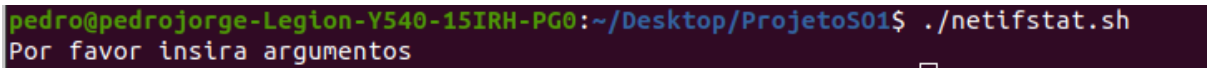


Fig. 24 - ./netifstat.sh

./netifstat.sh benfica - tentar introduzir um nome como argumento em vez de número

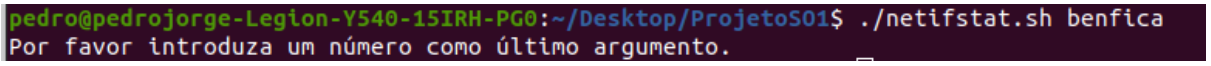


Fig. 25 - ./netifstat.sh benfica

./netifstat.sh -ç - tentar introduzir um argumento inválido, que não foi especificado

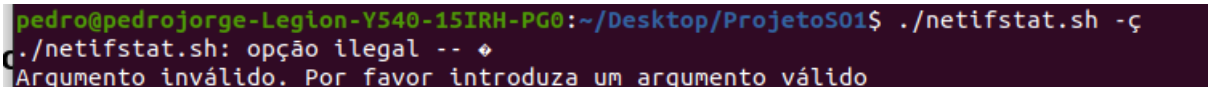


Fig. 26 - ./netifstat.sh -ç

./netifstat.sh -c "w.*" projeto - mesmo passando um argumento válido à opção -c, tentar introduzir um nome em vez dos segundos

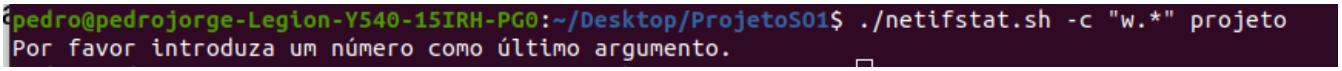


Fig. 27 - ./netifstat.sh -c "w.*" projeto

./netifstat.sh -p 5 10 - tentar mostrar um número de interfaces maior que o número de interfaces disponíveis.

Fig. 28 - ./netifstat.sh -p 5 10

./netifstat.sh -v -t - tentar ordenar de forma reversa o Tx Rate mas sem especificar os segundos

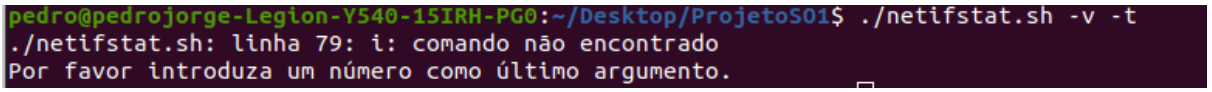


Fig. 29 - ./netifstat.sh -v -t

./netifstat.sh -c -R 10 - tentar pesquisar interface pela expressão regular e ordenar pelo RxRate mas sem especificar a expressão regular

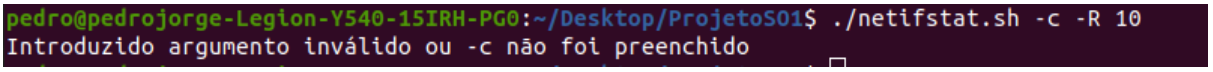


Fig. 30 - ./netifstat.sh -c -R 10

Conclusão

Em modo de conclusão, é possível afirmar que, de maneira geral, a resolução do projeto foi bem sucedida, dado que os resultados coincidem com o expectável.

Encontrei algumas dificuldades pelo caminho, no entanto, penso que o objetivo geral foi conseguido e que a resolução deste projeto enriqueceu o meu conhecimento sobre bash e aumentou a curiosidade sobre o mesmo.