## 

# Sistemas Operativos

### Professor Nuno Lau

### Departamento de Eletrónica, Telecomunicações e Informática

## 

## 

## Monitorização de interfaces de rede em bash

#### Ana Raquel Paradinha 102491

#### Paulo Pinto 103234

#### A participação na realização do trabalho foi igualitária.

# Índice

[**Índice**](#_ejp83own0ep9) **2**

[**1 Introdução**](#_dtt3rpdwxn06) **3**

[**2 Organização do código**](#_3q8uffqr12i5) **4**

[2.1 Declaração de variáveis globais](#_4c6zlgyzxwzh) 4

[2.2 Tratamento de argumentos](#_waym2ngax3v0) 5

[2.2.1 Função options()](#_in9cw57nyqob) 6

[2.2.2 Função error\_exit()](#_fsjbr5w68a2q) 6

[2.2.3 Função unit\_exit()](#_jfbz1oxe1zz9) 6

[2.2.4 Função sort\_exit()](#_p854rgf69d5q) 7

[2.2.5 getops](#_x5jezrtqy2mw) 7

[**4 Resultados**](#_xdx3zdq6cn46) **14**

[**5 Conclusão**](#_7iycrhtw3248) **20**

[**6 Bibliografia**](#_3g3ikf8dy0pe) **21**

# 1 Introdução

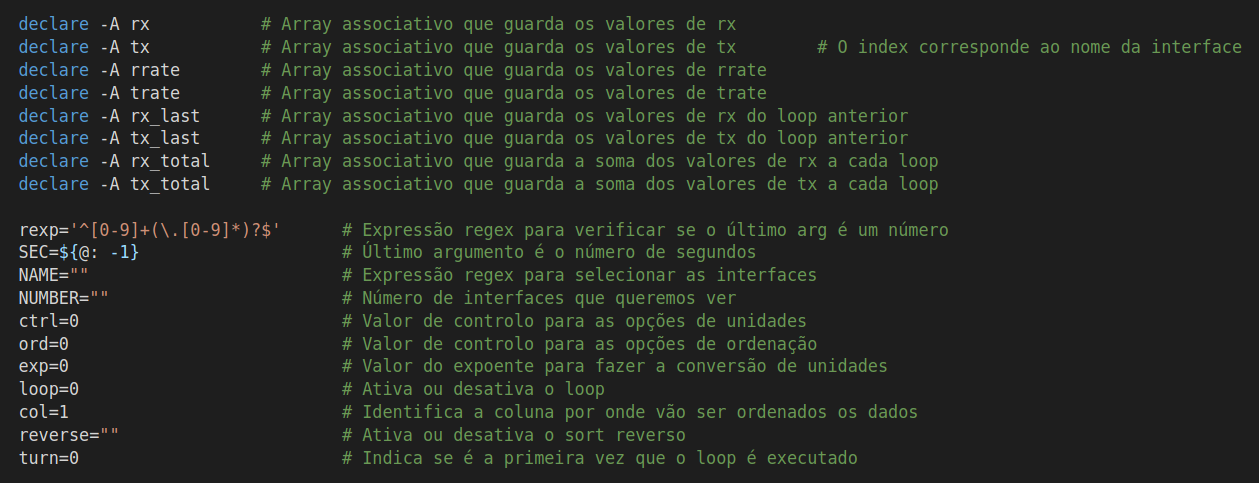
No âmbito da unidade curricular Sistemas Operativos foi-nos proposto o desenvolvimento do *script netifstat.sh* em *bash* que permite analisar a quantidade de dados transmitidos e recebidos, assim como as respetivas taxas de transferência, por cada interface de rede ativa na nossa máquina.

Com efeito, com base nos conteúdos explorados nas aulas práticas e alguma pesquisa em fontes externas conseguimos chegar ao resultado esperado. Nesse sentido, no presente relatório descrevemos as metodologias utilizadas para chegar a uma solução, assim como as dificuldades que encontramos e como foram resolvidas.

# 

# 2 Organização do código

## 2.1 Declaração de variáveis globais



Figura

Para o tratamento de dados, decidimos usar *arrays* associativos uma vez que nos permitem guardar valores associados a uma chave, a qual no nosso caso corresponde ao nome de cada interface de rede analisada.

Assim, definimos os seguintes *arrays*:

* **tx**, **rx** → guardam a quantidade de dados transmitidos e recebidos em cada interface de rede;
* **trate**, **rrate** → guardam as taxas de transferência para os períodos de tempo indicados ao executar o *script*;
* **tx\_last**, **rx\_last** → são usados quando a opção -l (loop) está ativa para passar os valores da quantidade de dados da segunda leitura do loop anterior para a primeira leitura do loop atual;
* **tx\_last**, **rx\_last** → guardam a soma cumulativa dos valores de tx e rx a cada loop.

Além disso, definimos uma expressão regular (**rexp**) para fazer a verificação de que o último argumento passado corresponde à quantidade de segundos que se pretende analisar, sendo que esta pode ser um número positivo inteiro ou decimal.

De seguida, inicializamos três variáveis dependentes dos valores passados como argumentos no terminal:

* **SEC** → número de segundos para análise, corresponde ao último argumento;
* **NAME** → corresponde à expressão regular pela qual vão ser filtrados os nomes das interfaces de rede a apresentar, só é definida quando a opção -c é passada como argumento;
* **NUMBER** → indica quantas interfaces vão ser apresentadas, só é definida quando a opção -p é passada como argumento.

Para efeitos de controlo das opções da unidade de visualização (-b, -k, -m) e de ordenação (-t, -r, -T, -R), uma vez que só pode ser passada uma de cada, definimos as variáveis **ctrl** e **ord**, respetivamente. Assim, estas variáveis são inicializadas a 0 (zero) e passam a 1 quando uma das opções que controlam é passada como argumento.

Seguidamente, temos a variável **exp** cujo valor vai ser usado como expoente para calcular a conversão de unidades, podendo ser 0, 1 ou 2 para bytes, kilobytes e megabytes, respetivamente.

Também definimos a variável **loop** que é ativada quando a opção -l é passada ao *script*.

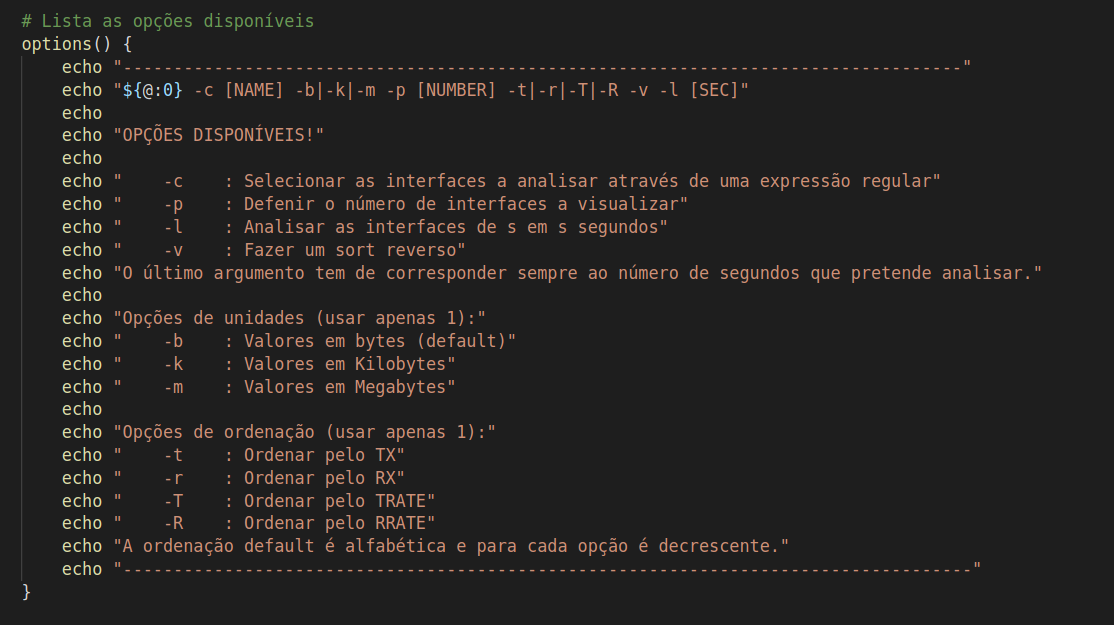
A nível das opções de ordenação temos ainda a variável **col** que identifica a coluna da tabela pela qual vai ser feita a ordenação e variável **reverse** que ativa/desativa o *sort* reverso. Esta última corresponde a uma *null string* quando a opção -v não está ativa e passa para “r” quando é ativada.

Por fim, a variável **turn** é usada quando a opção -l está ativa para indicar se o loop está na primeira iteração (valor 0) ou não (valor 1).

## 2.2 Tratamento de argumentos

Para o tratamento de argumentos foi utilizado o comando *getops,* que nos permite registar vários argumentos. Se esses argumentos forem válidos e estiverem corretamente aplicados, o comando *getops* vai proceder ao tratamento destes e realizar as tarefas que correspondem aos argumentos.

### 2.2.1 Função *options()*



Figura

A função ***options*** é utilizada para descrever como deve ser executado o *script netifstat.sh*, indicando as opções disponíveis, assim como a sua funcionalidade, e quais destas exigem argumentos.

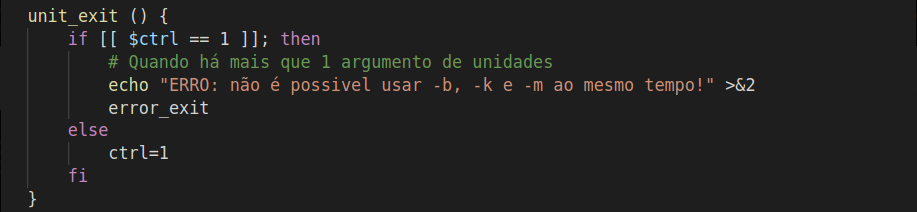
### 2.2.2 Função *error\_exit()*

## 

Figura

A função ***error\_exit*** é chamada sempre que um dos argumentos é usado de forma incorreta. Nesse sentido, executa a função *options* e termina o programa com o comando *exit 1*.

### 2.2.3 Função *unit\_exit()*

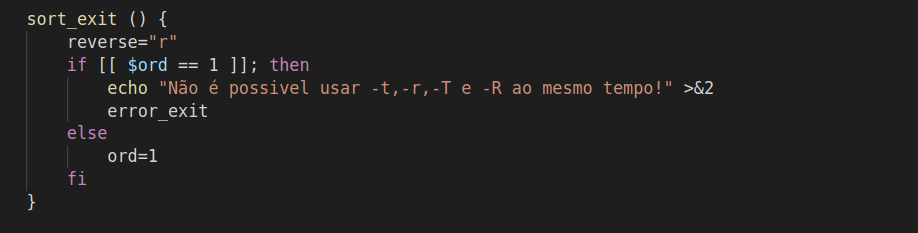


Figura

No caso dos argumentos *-*b*, -*k e/ou -*m* serem usados será chamada a função ***unit\_exit*** que averigua se estes não são usados em simultâneo. Fazendo uso da variável de controlo *ctrl,* se esta for igual a 0 significa que é a primeira vez que um deles é usado, permitindo assim continuar a verificação dos restantes argumentos e alterando o seu valor para 1.

Se a variável de controlo for igual a 1 quer dizer que já houve um argumento deste tipo a ser usado, levando a função a apresentar uma mensagem de erro relativa ao mau uso dos argumentos *-*b*, -*k e/ou *-*mseguida da execução da função *error\_exit.*

### 2.2.4 Função *sort\_exit()*

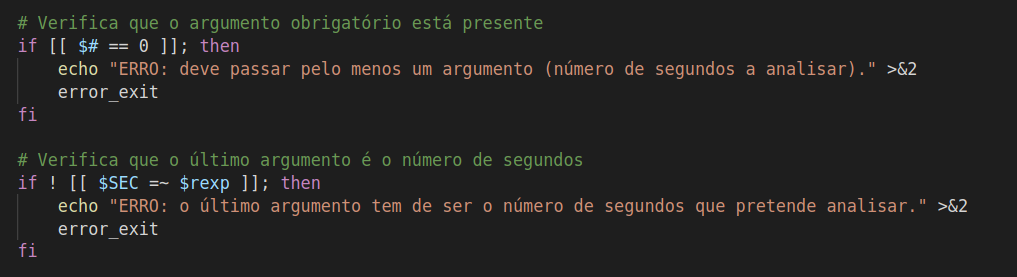


Figura

Para o tratamento das opções -t*,* -r*,* -Te -R é usada uma fórmula semelhante. Sempre que uma delas é usada a função ***sort\_exit*** é chamada definindo o valor de *reverse* como “r” e usando a variável de controlo *ord* verifica se alguma destas opções já foi utilizada antes.

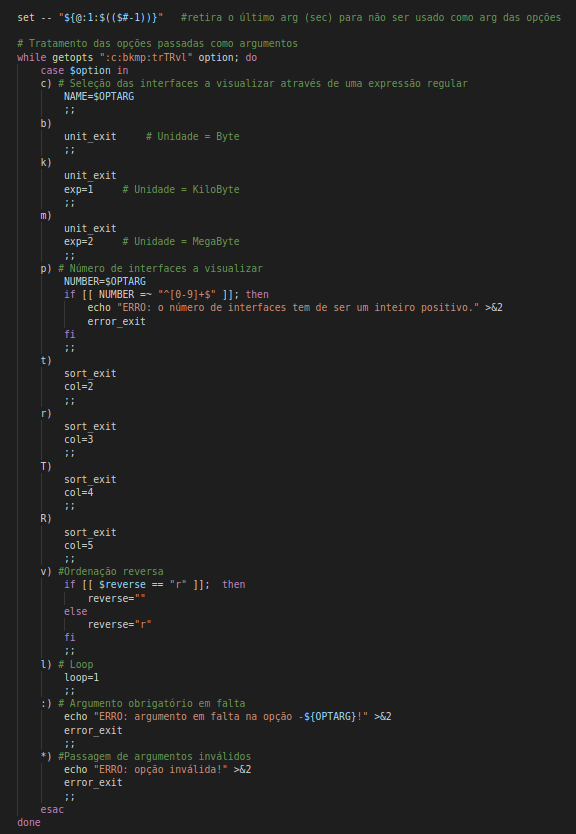
Se for a primeira vez a variável *ord* será definida como 1 e o tratamento de argumentos prossegue, se for a segunda vez, será impressa uma mensagem de erro relativa a estes argumentos e é chamada a função *error\_exit*.

### 2.2.5 *getops*



Figura

Antes de usar o comando *getops* são utilizados dois *if* que irão verificar se existe pelo menos um argumento passado e se o último argumento é um valor numérico inteiro ou decimal. Caso uma destas condições falhe é chamada a função *error\_exit (Fig.)* e impressa uma mensagem de erro.



Figura

Inicialmente, retiramos o último argumento, correspondente ao número de segundos estabelecido, do *array* predefinido *$@*, que agrupa todos os argumentos e opções passados na linha de comandos, com o objetivo de este não ser associado erradamente ao argumento de uma das opções definidas.

De seguida, usamos o comando *getopts* para tratar cada uma das opções passadas. Ao usar a expressão “:c:bkmp:trTRvl” definimos que o *getops* vai correr em modo *silent error checking* através do primeiro caráter “:”, o que nos permite definir o tratamento de erros que pretendermos.

Nesse sentido, quando a opção **-c** é verificada atribuímos o argumento (obrigatório) que lhe segue à variável *NAME*.

No caso das opções **-b**, **-k** e **-m** para além de ser executada a função *unit\_exit* é definido o valor de *exp* como 1 para -k e 2 para -m. Quando a opção é -b este valor não se altera, pois é o default.

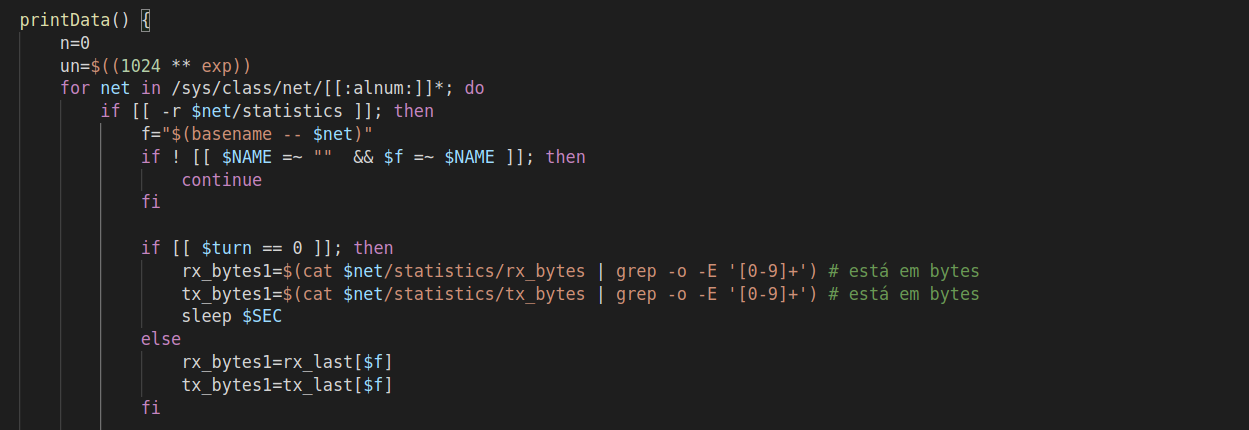
Para a opção **-p** o argumento que lhe segue (obrigatório) é atribuído à variável *NUMBER* e verifica-se se este é um inteiro positivo. No caso de não ser, é reportado um erro.

Adicionalmente, quando temos **-t**, **-r**, **-T** ou **-R**, para além da verificação já descrita (função *sort\_exit*) atribuímos os valores 2, 3, 4 e 5 à variável *col*, respetivamente.

Na opção **-v** o valor de *reverse* é alterado para “r” ou “ ” consoante o seu valor atual. Em **-l** apenas alteramos o valor de *loop* para 1.

Por fim, quando uma opção que exige argumento não o recebe, passa a constituir a variável $OPTARG da opção **“:”**, que desencadeará um erro. Além disso, se for passada uma opção não definida é executada a opção **“\*”** que relata esse erro.

3 Listagem de processos



Figura

A função ***printData*** é a principal do programa, pois é responsável por obter os valores necessários e imprimi-los.

Assim, a função começa por definir uma variável local **n** que serve para verificar quando o número de interfaces dado pela opção *-p* é atingido*,* caso esta seja selecionada*.*

Outra variável definida é ***un***que será usada para formatar os valores em bytes, kilobytes ou megabytes dependendo dos argumentos passados.

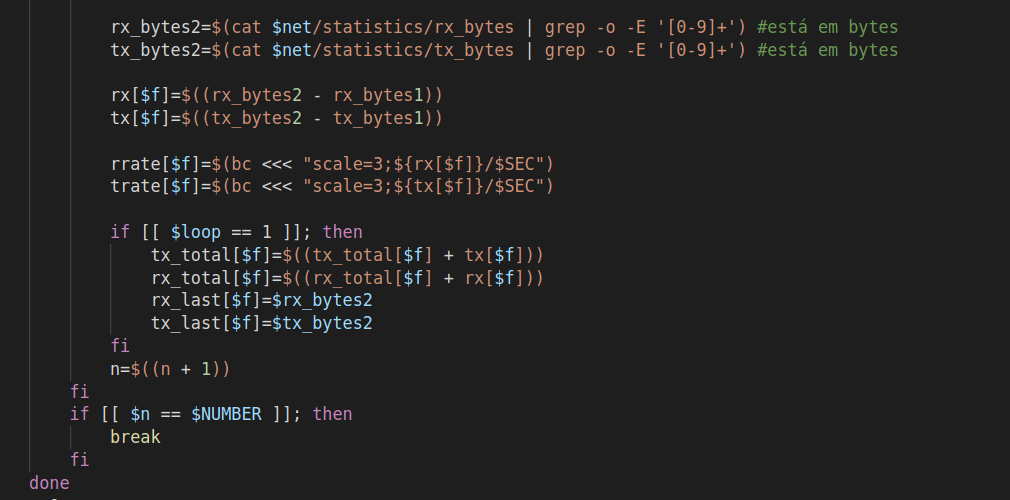
De seguida é criado um *for* que vai buscar o caminho da pasta de cada interface (por exemplo */sys/class/net/eth0*), no qual ***[[:alnum:]]\****indica que deve procurar por nomes alfanuméricos. O *if* posterior confirma a existência da pasta *statistics* dentro da pasta de cada interfacee só se tal acontecer é que o processo continua, caso contrário esta é ignorada e o ciclo *for* continua para a próxima interface.

No caso da pasta *statistics* existir, é definida uma variável *f* com o nome da interface através do comando *basename.* De seguida é criado um *if* que verifica se o nome da interface corresponde à expressão passada no argumento de-c, se tal não for verdade a interface é ignorada e o ciclo for passa para a seguinte. Caso o *-c* não seja definido este if é sempre falso, ou seja, todas as interfaces serão impressas.

O segundo *if* verifica se é a primeira vez que a função *printData* está a ser chamada. Este *if* é útil apenas quando o argumento *-l* é passado, o que leva a função a ocorrer em loop, enquanto que no resto dos casos ela apenas é chamada uma vez.

Para tal decisão é usada a variável de controlo *turn,* se esta for igual a 0 significa que é a primeira vez que a função *printData* é chamada e os primeiros valores de rx e tx são guardados nas variáveis ***rx\_bytes1*** e ***tx\_bytes1***, respetivamente. O comando ***cat*** serve para ir buscar o conteúdo da pasta especificada e o comando ***grep -o -E ‘[0-9]+’***confirma que apenas sejam guardados valores numéricos.

O *else* da condição só é executado quando a opção *loop* está ativa e tem por objetivo atribuir os valores guardados da análise anterior para serem usados como primeira medição da atual.



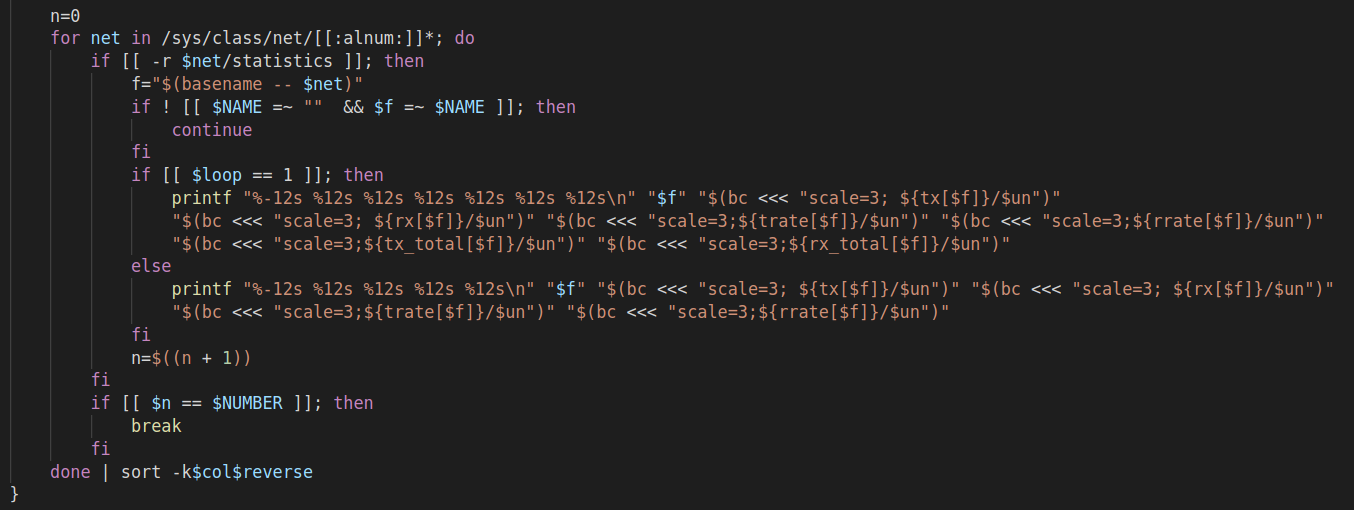
Figura

Em continuação, após o intervalo definido, determinamos os segundos valores de rx e tx pelo mesmo processo adotado para *rx\_bytes1* e *tx\_bytes1*. Com esses dados calculamos a diferença entre ambos para obter o tx e rx de cada interface de rede e adicionamo-los aos respetivos *arrays*.

De seguida, com o auxílio do comando *basic calculator* avaliamos o valor das taxas de transferência, **trate** e **rrate**, dividindo os valores de rx e tx da interface atual pelo intervalo de tempo definido.

O *if* que se segue só é executado quando a opção -l está ativa e nele incrementamos os **tx\_total** e **rx\_total** com os valores tx e rx da interface analisada em cada iteração do ciclo. Também atribuímos às variáveis ***tx\_last*** e ***rx\_last*** os valores de *tx\_bytes2* e *rx\_bytes2*, respetivamente, com o objetivo de ficarem acessíveis na execução seguinte do loop a fim de serem usadas como primeiro valor de tx e rx.

Além disso, a variável **n** é incrementada e comparada ao valor de *NUMBER*, quando a opção -p é passada no terminal, e termina o ciclo *for* quando estes são iguais, pois significa que foi atingido o número pretendido.



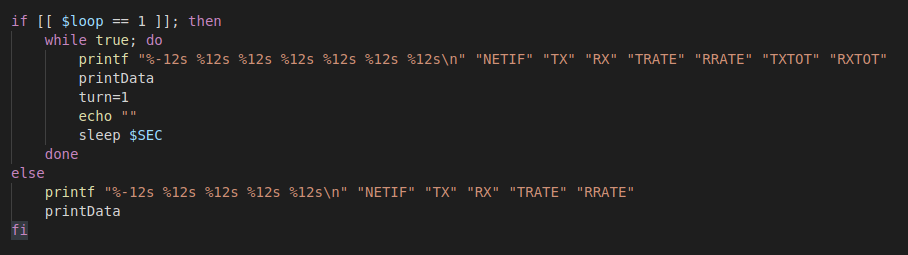
Figura

Seguidamente, o valor de **n** é novamente passado a zero, para ser utilizado num novo ciclo *for*.

As seis linhas de código seguintes são semelhantes à do ciclo anterior, cuja explicação já foi feita.

Nesse sentido, na condição *if* subsequente é executado o primeiro bloco caso a opção -l esteja ativa ou o segundo (*else*) caso contrário. Ambos convertem os dados obtidos às unidades especificadas pelas opções -b, -k ou -m, com recurso, novamente, ao comando bc e à variável *un*, definida anteriormente, e imprimem os mesmos numa tabela.

Os comandos até ao fim do ciclo *for* são uma repetição dos anteriores, sendo, no entanto, necessários para a execução do comando ***sort***. Com efeito, este recebe como argumento **-k** o valor de *col*, referente à coluna pela qual deve fazer a ordenação, e o de *reverse* que indica se será seguida a ordem default (crescente) para reverse igual a “ “ ou se será decrescente, com reverse a tomar o valor “r”.



Figura

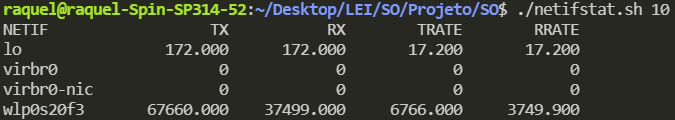
A última condição *if*, caso o *loop* esteja ativo, executa um ciclo *while* infinito, que imprime o cabeçalho da tabela com as duas colunas exclusivas desta opção (txtot e rxtot), chama a função *printData*, altera o valor de *turn* para 1, identificando que já foi feito um loop e corre o comando sleep para voltar a realizar uma medição de dados ao fim do tempo definido.

Por fim, caso se pretenda fazer a leitura uma única vez, é executado o bloco *else*, que apenas imprime o cabeçalho da tabela e corre a função *printData*.

# 4 Resultados

Nesta secção mostramos algumas das possíveis combinações de opções para executar o script *netifstat.sh*, as quais serviram também para verificarmos que eram obtidos os resultados esperados.

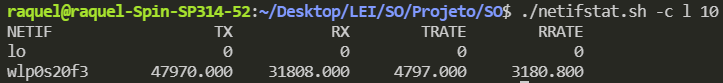
Por conseguinte, começamos pelo mais simples, passando apenas o número de segundos a analisar.



Figura

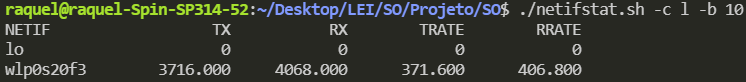
O resultado esperado era uma tabela com todas as interfaces de rede ativas ordenadas por ordem alfabética, o que se verifica.

Seguidamente, usamos a opção -c com a expressão regular “l”, pelo que se espera que apareçam apenas as interfaces que contêm este caráter no nome.

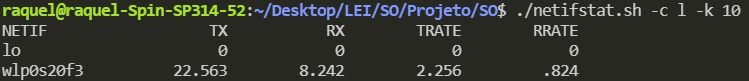


Figura

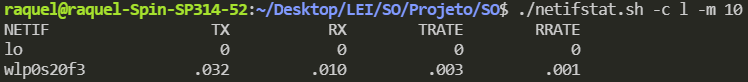
Os três testes seguintes avaliam as opções de conversão de unidades.



Figura



Figura



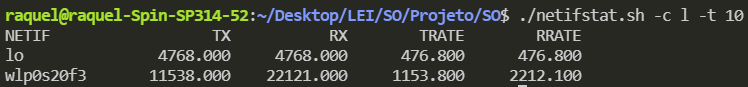
Figura

De seguida temos a opção -p com argumento igual a 1, portanto só queremos que apareça uma interface na tabela.

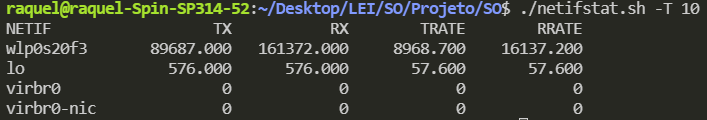


Figura

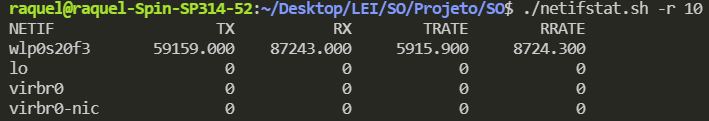
Para as opções de ordenação vamos obter respetivamente, sort (decrescente) pela coluna do TX, do TRATE, do RX e do RRATE.



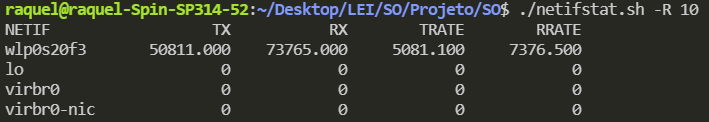
Figura



Figura

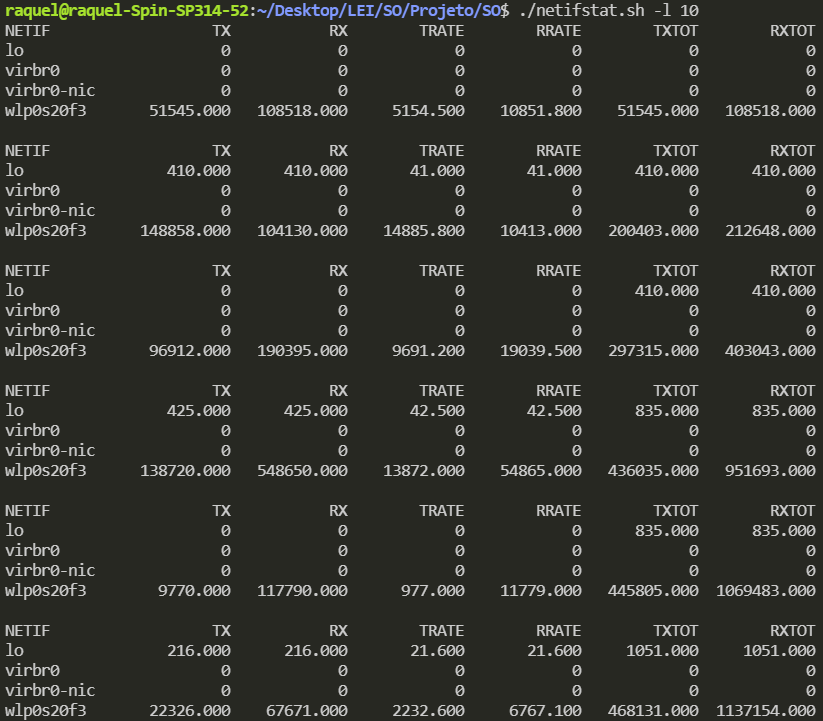


Figura



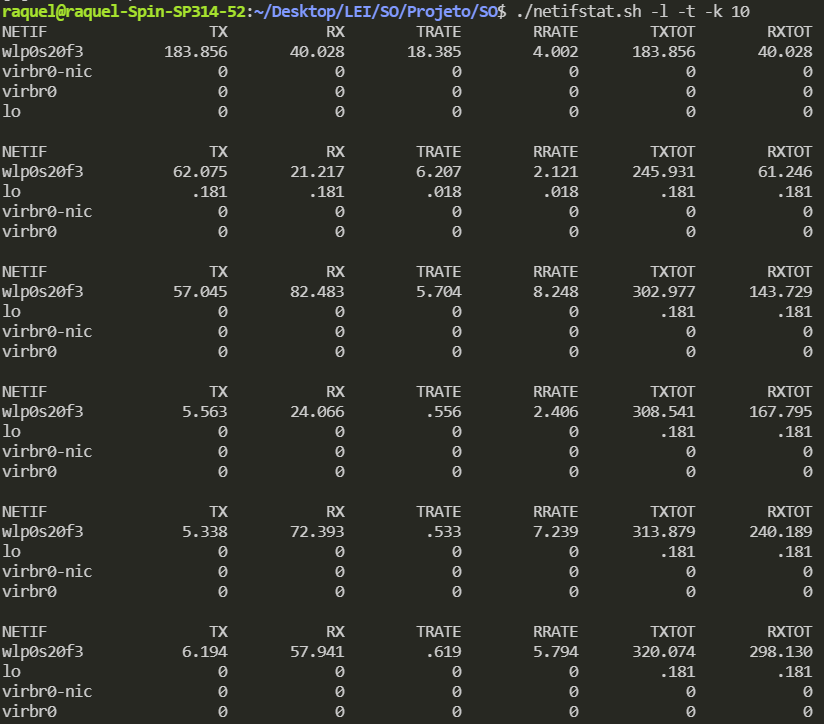
Figura

Seguidamente, verificamos o funcionamento da opção *loop*, que tal como esperado imprime a tabela de 10 em 10 segundos, atualizando os valores medidos.



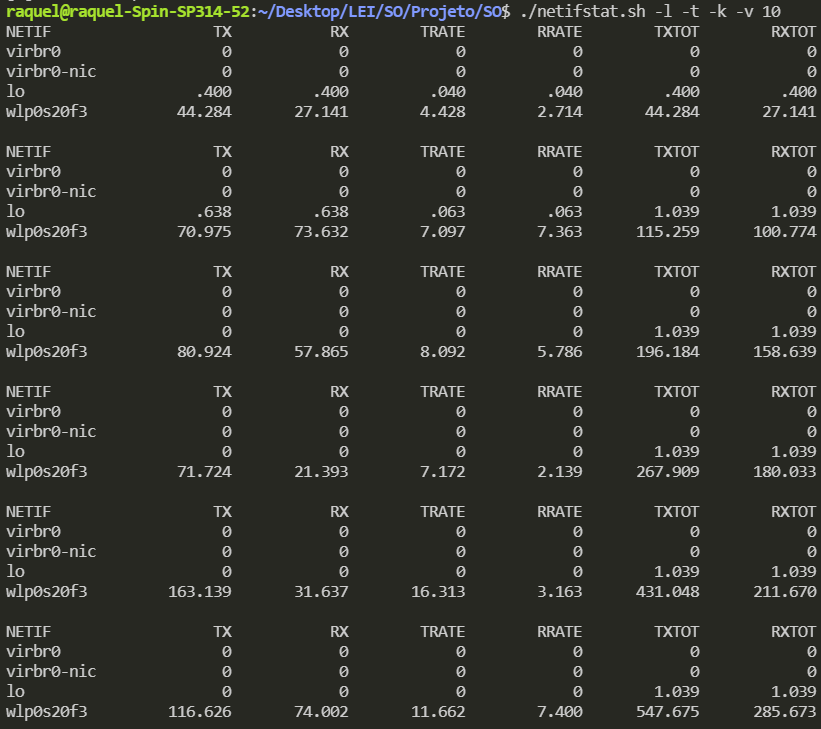
Figura

Testámos também a combinação entre as opções loop, ordenação pelo TX e unidades em kilobytes, obtendo o resultado esperado.



Figura

Por fim, repetimos o teste anterior adicionando a opção -v e confirmamos também o bom funcionamento da mesma.



Figura

# 5 Conclusão

O script desenvolvido vai de encontro ao proposto, permitindo analisar a quantidade de dados transmitidos e recebidos e as respetivas taxas de transferência, para além de ser possível utilizar diferentes filtros de visualização.

Com efeito, a realização deste trabalho contribuiu positivamente para o aprofundamento dos nossos conhecimentos em *bash*, através da pesquisa que exigiu, já que ficámos a conhecer novos comandos (p.ex. o *basic calculator*) e estruturas de dados, como os *arrays* associativos. Além disso, tivemos também mais contacto com a manipulação de ficheiros através da linha de comandos, aumentando a nossa experiência nesse campo.

Durante a execução do trabalho tivemos de lidar com algumas dificuldades, nomeadamente perceber como acederíamos aos valores desejados e qual a melhor maneira de os tratar, mas com recurso aos guiões práticos e alguns sites, nomeadamente fóruns de perguntas, conseguimos chegar ao resultado pretendido.

# 6 Bibliografia

Para a realização deste trabalho consultamos os slides teóricos e os guiões práticos disponibilizados pelo docente, assim como os seguintes sites (consultados entre os dias 20/11/21 e 05/12/21):

* <https://stackoverflow.com/>
* <https://www.computerhope.com/unix/bash/getopts.htm>
* <https://www.cyberciti.biz/faq/bash-get-basename-of-filename-or-directory-name/>
* <https://riptutorial.com/bash/example/31704/sort-by-keys>