Introdução

O script netistat.sh monitoriza as diferentes interfaces, calculando a quantidade de tráfico e velocidade que passa em cada uma delas num determinado intervalo de tempo.

Opções

Este programa, conforme as intenções do utilizador pode disponibilizar a informação das interfaces de várias formas, para isso terá que usar as várias opções disponíveis. A visualização por defeito estará formatada numa tabela com um cabeçalho seguido das interfaces organizadas por ordem alfabética.

opção -c

Esta opção será seguida de uma string que será um regex utilizado para selecionar a interface de rede.

opção -b

Visualização em bytes

opção -m

Visualização em megabytes

opção -l

Correr o programa em loop de s segundos

opção -t

Ordenar interfaces por ordem de bytes transmitidos TX

opção -r

Ordenar interfaces por ordem de bytes recebidos RX

opção -R

Ordenar interfaces por frequência de bytes RRATE mb/s ou b/s

opção -T

Ordenar interfaces por frequência de receção TRATE mb/s ou b/s

opção -v

Ordenar pela ordem contrária

Intenções inicias e workarrounds

A solução ideal para este problema incluiria um mapa ou um array bidimensional para armazenar os dados relativos a cada interface, mas como a bash não suporta essas estruturas nativamente, inicialmente pensamos em usar três arrays para armazenar os diferentes valores de rx, tx e as interfaces, o que funcionaria corretamente se não houvesse a possibilidade de ser adicionada uma ou mais interfaces a meio da execução programa (os índices das interfaces não iam corresponder aos índices dos valores). Então para resolver o problema declaramos uma estrutura a que chamamos de dicform.

```
declare -A dicform_i

dicform_i=()

declare -A dicform_f

dicform_f=()

declare -A dicform_r

dicform_r=()
```

Esta é semelhante a um dicionário em python ou um Map em Java, este é um array (dai -A) de strings que aceita como indices, strings.

Recolha de dados

Interfaces

Este é o ciclo usado para recolher a informação relativa ás interfaces

```
for inter in $(ifconfig -a | grep -E "inet|ether" -v | grep : | awk
```

```
'{print $1}' | cut -d ":" -f1); do

 interfaces+=($inter)

done
```

O **ifconfig** é um comando nativo dos sistemas Linux que dá nos a informações sobre interfaces de rede no sistema. Na informação disponibilizada pelo ifconfig todos os nomes relativos a interfaces têm uma coisa em comum, **":"** á frente do seu nome, então para chegarmos a elas redirecionamos a informação dando **pipe →|** para o comando **grep :** que seleciona as linhas com **":"**, no entanto com os nomes das interfaces vem outras informações, os endereços ipv6 pelo que antes temos de redirecionar a informação para um outro comando onde fazemos uma seleção inversa para tirar essas linhas **grep -E "inet|ether -v**.

Agora que temos as linhas onde a primeira coluna são interfaces podemos dar **pipe** para o comando **awk** e o argumento **{print $1}**, **awk é uma linguagem para processamento de texto inbutida na shell**, e aqui o estamos a fazer é só passar a informação redirecionada como argumento para o comando de **awk**, **print $1** que nos imprime a primeira coluna, depois disso apenas tiramos o dois pontos dando **pipe** para o comando **cut**.

Quando fazemos isto ficamos com uma coluna onde cada linha é uma interface, passando este output para uma variável, obtemos um array com as interfaces que depois percorremos no com o for adicionando á variável interfaces.

TX e RX

Ao todo temos 3 dicionários, 2 que armazenaram as informações no inicio e no fim de cada intervalo ,**dicform_i e dicform_f**, e outro as informações desde que corremos o programa para as situações de loop, **dicform_r** .

Sendo que obtemos as informações para essas estruturas com os seguintes comandos:

for i in "${interfaces[@]}"; do

dicform[$i]=$(ifconfig $i | grep packets | grep TX | awk '{print $5}')

dicform[$i]+=":"$(ifconfig $i | grep packets | grep RX | awk '{print $5}')

done

Para obter a informação de uma única interface corremos o ifconfig com o nome da interface á frente , **ifconfig $i**, pelo que aqui percorremos cada interface para buscar os valores de **tx** e **rx**.

Como na recolha das interfaces usamos o **grep** e **awk** para ir buscar a informação que nos interessa. Primeiro selecionamos todas as linhas que têm **packets** , isto porque se utilizasse-mos **bytes** e houvesse queda de pacotes iriam aparecer colunas extra.

Com a informação filtrada vamos novamente fazer uma seleção, onde selecionamos TX e RX buscando os bytes e com o **awk** imprimindo a quinta coluna e concatenando-a uma string com uma formatação especifica para posterior fácil acesso que guardamos nos dicionários pretendidos.

Sendo essa formatação a seguinte “tx:rx”

Funções e blocos

O programa está dividido em varias funções e blocos de codico que se interligam entre si.

assert()

Verifica se o numero de argumentos é valido e se o argumento passado como segundos é um numero inteiro

```
assert() { #verifica a quantidade de argumentos

 re='^[0-9]+$'

 if ! [[ $s =~ $re ]]; then # se o argumento do tempo nao for um numero e o numero total de argumentos nao for igual ou superior a 2 o programa nao corre

 if [[ $n_arg -lt 2 ]]; then

 echo "Assertion failed: parâmetro obrigatório em falta (Tempo em segundos )"

 exit $E_ASSERT_FAILED
```

```
 fi

 fi

 if [ $n_arg -le 0 ]; then

 echo "Assertion failed: parâmetro obrigatório em falta (Tempo em
segundos )"

 exit $E_ASSERT_FAILED

 fi

}
```

getinterfaces()

Reúne num array todas as interfaces no sistema

```
function getinterfaces() {

 for inter in $(ifconfig -a | grep -E "inet|ether" -v | grep : | awk
'{print $1}' | cut -d ":" -f1); do

 interfaces+=($inter)

 done

}
```

getdicform()

Obtém os dados mais recentes fornecidos pelo ifconfig para o **dicform_f**

```
function getdicform() {
```

```
 for i in "${interfaces[@]}"; do

 dicform_f[$i]=$(ifconfig $i | sort | grep packets | grep TX | awk
'{print $5}')

 dicform_f[$i]+=":"$(ifconfig $i | sort | grep packets | grep RX | awk
'{print $5}')

 done

}
```

ciclo getops

Este lê as opções introduzidas pelo utilizador até ao primeiro argumento não opção. Usa uma string que começa por “:” onde cada letra é uma opção e se tiver “:” á sua frente têm um argumento que irá ser lido.

Dentro deste ciclo são definidas as opções **order_of_sort** para o **sort** que vai receber o pipe do **print_dados()**. São atribuídos valores aos atributos **loop** que nos indica se existe loop ou não; **opv** que nos indica o tipo de dados (0-b;1-kb;2-mb); **grepby** e **c_option** que indicam ao programa para dar pipe do print_dados() para o o grep seguido do regex da opção c , **reverse** que inverte a ordem anterior estabelecida (reverse=1) e ao **head** que é argumento e comando a que damos o ultimo pipe que nos limita o numero de linhas do output e por sua vez o numero de interfaces.

```
while getopts bc:lmp:rRtTvk option ; do

 case $option in

 c)

 c_option="true"

 grepby=$OPTARG

 ;;
```

```
 k)  opv=2;;

 l)  loop=1;;

 m)  opv=1;;

 p)  n_head="-"$(echo $OPTARG);;

 r)  order_of_sort="-k 3 -n";;

 R)  order_of_sort="-k 5 -n";;

 t)  order_of_sort="-k 2 -r";;  ## so here is -r to make it iqual to
the pdf example

 T)  order_of_sort="-k 4 -n";;

 v)  reverse=1;;

 esac

done
```

print_dados()

Aqui é onde o output é formado.

Começamos por chamar a função getdicform para obter os dados mais recentes.

Dentro do ciclo obtemos o output relativo a cada interface $i, primeiro obtemos o linhas dos dicionários inicial , final , e desde do startup.

A partir dessas linhas obtemos o valor inicial e final de **rx** e **tx** ,**rx_i**,**tx_i**,**rx_f** e **tx_f**.

Subtraindo os valores inicias e finais obtemos o **rx** e **tx** a imprimir.

Com **rx** e **tx**, dividimos estes pelo tempo **$s** defendido pelo utilizador, dando pipe ao comando **bc**,que recebendo pipe de uma string com calculo e certeza calcula o resultado.com casas decimais, e obtemos os valores relativos á taxa de transmissão e taxa de receção (Trate,Rrate) , **rr** e **tr**

Além destes calculamos o valor total caso se trate de um loop subtraindo os valores atuais aos que estavam no dicionário dicform_r ,**rx_f - r_inicial**, sendo estes os valores totais de bytes de rx e tx, **tx_total**,**rx_total**.

Ainda dentro do ciclo apartir da variável **opv** fazemos um switch case onde conforme o valor calculamos o valor a imprimir em bytes , kb , mb e dentro desse case temos dois prints possíveis conforme o valor de **loop** , se **loop** for 1 fazemos o print com os valores totais se não fazemos o print normal.

```
function print_dados() {

 getdicform

 for i in "${interfaces[@]}"; do

 linha_i="$i ${dicform_i[$i]}"

 linha_f="$i ${dicform_f[$i]}"

 linha_r="$i ${dicform_r[$i]}"

 rx_i=$(echo $linha_i | grep $i | awk '{print $2}' | cut --complement -
d ":" -f1)

 tx_i=$(echo $linha_i | grep $i | awk '{print $2}' | cut -d ":" -f1)

 r_inicial=$(echo $linha_r | grep $i | awk '{print $2}' | cut --
complement -d ":" -f1)

 t_inicial=$(echo $linha_r | grep $i | awk '{print $2}' | cut -d ":" -
f1)

 rx_f=$(echo $linha_f | grep $i | awk '{print $2}' | cut --complement -
d ":" -f1)

 tx_f=$(echo $linha_f | grep $i | awk '{print $2}' | cut -d ":" -f1)

 let rx=rx_f-rx_i

 let tx=tx_f-tx_i
```

```
 let t_total=t_inicial+tx

 let r_total=r_inicial+rx

 rr=$( bc <<< "scale=2; $rx / $s" )

 tr=$( bc <<< "scale=2; $tx / $s" )

 case $opv in

 0)

 if [[ loop -ne 0 ]]; then

 printf "%-10s\t%10s\t%10s\t%10s\t%10s\t%10s\t%10s\n" "$i" "$tx B" "$rx
B" "$tr B/s" "$rr B/s" "$t_total B" "$r_total B"

 else

 printf "%-10s\t%10s\t%10s\t%10s\t%10s\n" "$i" "$tx B" "$rx B" "$tr
B/s" "$rr B/s"

 fi

 ;;

 1)

 rx=$(bc <<<"scale=2;$rx/1000000")

 tx=$(bc <<<"scale=2;$tx/1000000")

 rr=$(bc <<<"scale=2;$rr/1000000")

 tr=$(bc <<<"scale=2;$tr/1000000")

 if [[ $loop -ne 0 ]]; then

 printf "%-10s\t%10s\t%10s\t%10s\t%10s\t%10s\t%10s\n" "$i" "$tx MbB"
"$rx Mb" "$tr Mb/s" "$rr Mb/s" "$t_total Mb" "$r_total Mb"
```

```
 else

 printf "%-10s\t%10s\t%10s\t%10s\t%10s\n" "$i" "$tx Mb" "$rx Mb" "$tx
Mb" "$rr Mb"

 fi

 ;;

 2)

 rx=$(bc <<<"scale=2;$rx/1000")

 tx=$(bc <<<"scale=2;$tx/1000")

 rr=$(bc <<<"scale=2;$rr/1000")

 tr=$(bc <<<"scale=2;$tr/1000")

 if [[ $loop -ne 0 ]]; then

 printf "%-10s\t%10s\t%10s\t%10s\t%10s\t%10s\n" "$i" "$tx Kb"
"$rx Kb" "$tr Kb/s" "$rr Kb/s" "$t_total Kb" "$r_total Kb"

 else

 printf "%-10s\t%10s\t%10s\t%10s\t%10s\n" "$i" "$tx Kb" "$rx Kb" "$tx
Kb" "$rr Kb"

 fi

 esac

 done

}
```

Initial_setup()

Como o nome diz é uma função unicamente usado no inicio da execução do programa para introduzir os dados no dicionário inicial (dicform_i) e dicionario com os dados desde do startup (dicform_r), além disso é imprimido o primeiro cabeçalho e é executado o comando sleep $s em que $s é o intervalo de tempo.

```
function inicial_setup(){ #Setup inicial

 getinterfaces

 for i in "${interfaces[@]}"; do

 dicform_i[$i]=$(ifconfig $i | sort | grep packets | grep TX | awk
'{print $5}')

 dicform_i[$i]+=":"$(ifconfig $i | sort | grep packets | grep RX | awk
'{print $5}')

 dicform_r[$i]=$(ifconfig $i | sort | grep packets | grep TX | awk
'{print $5}')

 dicform_r[$i]+=":"$(ifconfig $i | sort | grep packets | grep RX | awk
'{print $5}')

 done

 if [[ $loop -eq 0 ]] ; then

 printf "%-10s\t%10s\t%10s\t%10s\t%10s\n\n" "NETIF" "TX" "RX" "TRATE"
"RRATE"

 else

 printf "%-10s\t%10s\t%10s\t%10s\t%10s\t%10s\t%10s\n\n" "NETIF" "TX"
"RX" "TRATE" "RRATE" "TXTOT" "RXTOT"

 fi

 sleep $s
```

```
}
```

the_real_thing

Este nome foi o que demos á ultima secção de código pois é nela que tudo acontece á parte do processamento de opções e o **initial_setup()**, aqui é nos apresentado uma condição que verifica se existe **loop**, e se tal corre um ciclo infinito onde são imprimidos os dados e redefinidos valores iniciais a cada iteração , se não imprime uma única vez os dados. Em ambas as opções estes dados são passados por múltiplos pipes conforme as opções defenidas no **getops**.

```
inicial_setup

if [[ $loop -eq 1 ]]; then # parte do loop caso seja selecionada a opção -l

 while true ; do

 if [[ $c_option == "true" ]]

 then

 print_dados |  sort $order_of_sort |grep $grepby | head $n_head # parte não loop

 else

 print_dados |  sort $order_of_sort | head $n_head # parte não loop

 fi

 for i in "${interfaces[@]}"; do

 dicform_i[$i]=$(ifconfig $i | sort | grep packets | grep TX | awk '{print $5}')

 dicform_i[$i]+=":"$(ifconfig $i | sort | grep packets | grep RX | awk '{print $5}')
```

```
 done

 sleep $s

 echo "------------------------------------------------------------------
----------------------------------------"

 printf "%-10s\t%10s\t%10s\t%10s\t%10s\t%10s\t%10s\n\n" "NETIF" "TX"
"RX" "TRATE" "RRATE" "TXTOT" "RXTOT"

 done

else

 if [[ $c_option == "true" ]]

 then

 print_dados |  sort $order_of_sort  |grep $grepby | head $n_head #
parte não loop

 else

 print_dados |  sort $order_of_sort | head $n_head # parte não loop

 fi

fi
```

Fluxo

A execução de funções e blocos de códico pode ser abstractamente representada por este “fluxograma”

assert() → getops → initial_setup() → the_real_thing → print_dados()^n , n€[1,+infinito]

Testes feitos

Normal

./netistat.sh 4

Loop/Kb

./netistat.sh -l -k 4

Interfaces/Mb/tx/regex

./netistat.sh -p 3 -c regex -m -t 4

Interfaces/b/p/reverse

./netistat.sh -b -p 3 -v 4

Conclusão

Com este trabalho ficamos com uma melhor compreensão de como a bash shell apesar de uma linguagem simples sem estruturas complexas e uma sintaxe única com indentações por vezes irregulares ao contrário da maioria das linguagens de programação, é uma ferramenta poderosa para interagir com sistema operativo capaz de processar outputs e inputs de diferentes programas e comandos selecionado e transformando-os com as diferentes opções possíveis sendo perfeita para interagir com outras linguagens executando binários e processando os resultados para decidir o que fazer.

Trabalho realizado por Tiago Portugal e Airton Moreira