Sistemas Operativos 2019/20

Primeiro Trabalho Prático

Estatísticas de Utilizadores em Bash

27 de Novembro de 2019

Trabalho realizado por :

- Daniel Gomes,93015
- André Morais,93236

Índice

Índice 1

Introdução 2

Userstats.sh 3
- Variáveis Globais 3
- Funções 4
- Processamento De Opções 11
- Main 16
- Testes Efetuados 16

Comparestats.sh 19
- Variáveis Globais 19
- Funções 20
- Processamento de Opções 22
- Main 23
- Testes Efetuados 25

Conclusão 27

Introdução

Com este trabalho foi nos proposto desenvolver scripts de forma a recolher informações sobre como os utilizadores estão a usar o sistema computacional, nomeadamente o número de sessões de cada um e o respetivo tempo associado a estas. Além disso, outro objetivo deste trabalho foi a capacidade de comparar os dados conseguidos em períodos distintos. Assim, tendo como base cada um destes objetivos, criamos os scripts *userstats.sh* e *comparestats.sh*.

Userstats.sh

Com vista a realização do primeiro objetivo deste trabalho,criou-se o script ***Userstats.sh***, pelo que, com vista a uma melhor organização e compreensão do código,decidiu-se estruturar este no seguinte formato:

- Variáveis Globais
- Funções
- Processamento das Opções
- *Main*

Todo este processo baseia-se no comando de Linux , ***last*** , que mostra as sessões de cada utilizador do computador mais recentes.

Variáveis Globais

```
#-----Variáveis globais-----
file="/var/log/wtmp"
a="last"
userstatsarray=()
indexuserstats=0
varsort="sort"
argumentsarray=("$@")
flag=0
```

Dentro desta secção, inicialmente temos a variável ***file,*** que corresponde à *string* com o nome do ficheiro predefinido, de onde o comando last recolhe os dados necessários para imprimir no terminal aquilo que já foi referido anteriormente.

Seguidamente,declaramos a variável ***a*** , que contém o nome do comando ***last,***que irá servir posteriormente para vários motivos,nomeadamente para o tratamento/processamento de opções inseridas pelo utilizador do *script*.

Depois, introduzimos um *array* (estrutura de dados), denominado de ***userstatsarray***, que irá conter os dados já processados, sendo assim algo necessário para conseguir imprimir todos estes no terminal. Com vista à “indexação” da estrutura de dados referida anteriormente, gerou-se então a variável ***indexuserstats***.

Para podermos gerar as opções de ordenação, introduzimos a variável ***varsort*** que contém a string “**sort**”, e posteriormente, de forma a que pudéssemos verificar quais as opções de ordenação utilizadas, utilizamos uma *flag ,* que se apresenta como uma variável booleana.

Por fim neste excerto de código, ***argumentsarray*** é a nossa estrutura de dados que contém todos os argumentos introduzidos ao correr o programa,que é

conseguido através de `$@` que na *Bash*, é nada mais nada menos, que os parâmetros passados ao script.

Funções

```
#-------Funçoes-----------------
function usage (){
   echo "Options Error! "
   echo " Script Usage:"
   echo '       OPTION                  Description                 '
   echo '      -s "date":      Only shows sessions after "date". Put a
correct date format!'
   echo '      -e "date":      Only shows sessions before "date"'
   echo '      -u "regex":    If the  "regex" matches the sessions,
those will be displayed '
   echo '      -g "user group":Only shows sessions which belong to the
users group "group"'
   echo '      -f "file":      Shows sessions only "file" '
   echo ' Sorting : '
   echo "          -r:             Sort all data in reverse order"
   echo "          You can not use more than 1 option from these:"
   echo ' '
   echo "          -n:             Sort by number of sessions"
   echo "          -t:             Sort by total logged time"
   echo "          -a:             Sort by maximum logged time"
   echo "          -i:             Sort by minimum logged time"
}
```

A função ***usage()*** , é utilizada neste script como forma de mostrar ao utilizador como usar este script , imprimindo no terminal todas as opções disponíveis, e , como consequência, mostra a funcionalidade de cada uma delas. Esta função será apenas chamada no processamento de opções , e apenas na possibilidade de ser introduzida uma opção não válida.

```
function getusers(){
column='$1'
findcolumn="| awk '{print $column}'"
optionsarr="${a} $findcolumn | sort |uniq"
userarr=($(eval $optionsarr))
}
```

Dentro da função ***getusers()***, o código existente tem em vista guardar os utilizadores únicos das sessões presentes no comando ***last***, dentro de um array. Para que isto fosse possível concatenamos a variável ***a*** *(*cujo intuito foi já referido anteriormente), com o conteúdo de ***findcolumn***, e ,por sua vez, com os comandos ***sort*** e ***uniq.*** Assim, ***findcolumn*** guarda em si as funcionalidades do comando ***awk*** , bastante efetivo na impressão de frases na forma de padrões que serão procurados em cada linha de um documento, pelo que neste caso, procuramos imprimir a primeira coluna, ***$1***, guardada na variável ***column***. Seguidamente, o comando sort e uniq, juntos, irão ordenar os padrões únicos encontrados daquilo que contém ***a*** e ***findcolumn*** concatenados. Para guardarmos o processo todo referido num array denominado ***userarr*** , recorremos ao comando ***eval,*** para que seja possível executar o conteúdo de ***optionsarr*** que por sua vez é inserido na estrutura de dados ***userarr***. Como eval executa tudo na forma de comando tivemos de guardar em variáveis alguns conteúdos de forma a não ler,por exemplo, “***$1”***, como um comando mas sim como ***string*** .

```
function gettime(){
	tottime=0
	maxtime=0
	session=0
	mintime=0
	firsthour=${timearr[0]}
	firsthour=$(echo ${firsthour:1:2}| awk  '{sub(/^0*/,"");}1')
	firsthour=$(echo ${firstminute}| tr --delete +)
	firstminute=${timearr[0]}
	firstminute=$(echo ${firstminute:4:2}| awk
'{sub(/^0*/,"");}1')
	firstminute=$(echo ${firstminute}| tr --delete :)
	let "mintime= firstminute + 60*firsthour"
```

A função ***gettime(),*** é consideravelmente extensa pelo que a descrição desta neste relatório, será feita por partes. Inicialmente, são definidas e inicializadas as variáveis tottime,maxtime,session e mintime, que respetivamente, correspondem ao tempo total de cada utilizador, o tempo máximo, o tempo de cada de sessão e o tempo mínimo de cada sessão. Depois de inicializadas, as variáveis utilizadas ***firsthour*** e ***firstminute*** são utilizada para calcular o tempo da primeira sessão de cada utilizador (que será o nosso tempo mínimo inicialmente) , onde recorre-se a um "*regex*" dentro do comando ***awk*** para retirar os zeros a esquerda, visto que em Bash todos os números começados por 0 (por exemplo *01,02…*),apresentam-se em base octal, e consecutivamente *08* e *09* seriam números inválidos. A utilização de ***tr -- delete*** foi útil para eliminar caracteres específicos caso fossem encontrados, que afetavam o uso desta função num teste realizado, que se apresentavam insignificantes.

```
for time in ${timearr[@]}
        do
            lenghttime=${#time}
            if [[ lenghttime -eq 7 ]]
            then
                let "session = 0"
                min=${time:4:2}
                hour=${time:1:2}
                hour=$(echo $hour | awk  '{sub(/^0*/,"");}1')
                min=$(echo $min | awk  '{sub(/^0*/,"");}1')
                let "session = min + 60*hour"   #tempo da sessao em
causa
                let "tottime += min + 60*hour"  #tempo total
                if [[ session -gt maxtime ]]
                then
                    let "maxtime = session"
                fi
                if [[ session -lt mintime ]]
                then
                    let "mintime = session"
                fi
```

De seguida, percorre-se todas as sessões efetuadas pelo utilizador nas condições definidas pelo utilizador ao introduzir opções(ou não), que estão contidas no array ***timearr*** (definido noutra função). Dentro do loop, o que é feito em cada condição if é exatamente o mesmo, ou seja, o cálculo de tempo total, mínimo e máximo, contudo a diferença nestas condições consiste no tamanho do de cada elemento do array do tempo: 7 se apenas existirem horas e minutos, 9 se existirem dias no intervalo [1,9] juntamente com horas e minutos, e , por fim, 10 se existirem dias no intervalo [10,99] juntamente com horas e minutos. Assim, foi descartada a hipótese de haverem sessões com mais de 99 dias de duração.

```
 elif [[ lenghttime -eq 9 ]]
           then
               let "session = 0"
               day=${time:1:1}
               min=${time:6:2}
               hour=${time:3:2}
               day=$(echo $hour | awk  '{sub(/^0*/,"");}1')
               hour=$(echo $hour | awk  '{sub(/^0*/,"");}1')
               min=$(echo $min | awk  '{sub(/^0*/,"");}1')
               let "session = min + 60*hour + day*24*60 "   #tempo da
sessao em causa
               let "tottime += min + 60*hour + day*24*60"  #tempo
total
               if [[ session -gt maxtime ]]
               then
                   let "maxtime = session"
               fi
               if [[ session -lt mintime ]]
               then
                   let "mintime = session"
               fi
       #caso haja uma sessao de tempo cujos numero de dias esteja
entre [10,100[
           elif [[ lenghttime -eq 10 ]]
           then
               let "session = 0"
               day=${time:1:2}
               min=${time:7:2}
               hour=${time:4:2}
```

```
                day=$(echo $hour | awk  '{sub(/^0*/,"");}1')
                hour=$(echo $hour | awk  '{sub(/^0*/,"");}1')
                min=$(echo $min | awk  '{sub(/^0*/,"");}1')
                let "session = min + 60*hour + day*24*60 "   #tempo da
sessao em causa
                let "tottime += min + 60*hour + day*24*60"  #tempo
total
                if [[ session -gt maxtime ]]
                then
                    let "maxtime = session"
                fi
                if [[ session -lt mintime ]]
                then
                    let "mintime = session"
                fi

            fi

        done
}
```

Assim, para cada utilizador, se o tempo total de uma determinada sessão for menor que o mínimo já guardado, este passará a ser o mínimo tempo registado. O contrário acontece para o tempo máximo registado.Somando na variável ***tottime*** , em cada iteração conseguimos de forma eficaz calcular o tempo total das sessões para a qual foram encontrados dados, tendo em conta, mais uma vez, as opções dos utilizadores.

Na função ***processdata()*** , apresentada na imagem a seguir, é onde ocorre todo o processamento dos dados recolhidos até então e também de outros futuros. O objetivo desta consiste em usar o resultado das funções definidas anteriormente, tanto a ***getusers()*** como a ***gettime()*** , para que se consiga apresentar o resultado final pretendido, que consiste na impressão dos dados.

```
function processdata(){
```

```
   getusers
       for element in ${userarr[@]}
       do
               if [[ $element != "reboot" && $element != "wtmp" &&
$element != "shutdown" ]]
           then
               numbersessions=$(last -f ${file} |grep -o $element | wc
-l)
               timearr=($(last -f ${file} | grep $element | awk '{if
(( $10 !~ /in/ && $10 !~ /running/ )) { print $10 }}'  | tr " " " " ))
               for arg in ${argumentsarray[@]}
               do
                 if [ $arg == "-s" ] || [ $arg == "-e" ]
                 then
                       timerr=$($a -f ${file} | grep $element | grep -
v 'no' | awk '{if (( $10 !~ /in/ && $10 !~ /running/ )) { print $10
}}'  | tr " " " ")
                       timearr=($timerr)
                       numbersessions=$($a -f ${file} |grep -o
$element | wc -l)
                       break
                 fi
               done
               gettime
   printsuserstats=$(echo $element $numbersessions $tottime $maxtime
$mintime)
             #adicionar ao array....
   userstatsarray[$indexuserstats]=$printsuserstats
   indexuserstats=$((indexuserstats+1))
           fi
       done
       echo "------"
       if [[ ${#userstatsarray[@]} -eq 0 ]]; then
           echo "Something went wrong.No Data Found!"
           exit 1
       fi
}
```

Para que seja possível concretizar a finalidade referida agora, itera-se sobre todos os utilizadores, que preenchem os requisitos necessários para aquilo que o utilizador pretender visualizar no terminal, excluindo logo de partida, para todo e qualquer caso, usernames denominados ***“reboot”, “wtmp” , ou “shutdown”*** .

Dentro do loop, a variável ***numbersessions*** contem com o número de sessões conseguidas para o utilizador em causa, algo que foi possível devido ao comando ***grep*** , que permite a procura de um determinado padrão ao longo do “ficheiro” , pelo que contando o número de vezes que cada nome de utilizador ocorre (***wc-l***), obtém-se o pretendido.

Além disto, tem-se a estrutura de dados ***timearr*** que consegue obter a duração de cada sessão do user em causa em cada iteração deste ciclo. Contudo, ocorreram alguns problemas na obtenção do tempo e número de sessões utilizando a variável ***a*** no caso de serem utilizadas as opções ***-s*** e ***-e*** (o seu uso será referido posteriormente), pelo que, como forma de solucionar este problema, quer o tempo, quer o número de sessões são inicialmente obtidas diretamente do comando ***last*** e nao da variável concatenada ***a*** logo, itera-se pelo número de argumentos e no caso haver um match com a opção -s ou -e, recorre-se de forma contrária a variável **a,** e finalmente ***timearr*** e ***numbersessions*** terão um novo valor associado parando este segundo ciclo com o uso de ***break.***

A função ***gettime()*** tem o seu uso respetivo ao ser chamada, e assim, recolhe se todos os dados necessários para guardar no array ***userstatsarray()***.

Terminando esta função, ocorre uma ligeira verificação: caso o array fique vazio, por não haverem dados associados, imprime-se uma mensagem de erro e termina-se o programa.

```
function verifyflag(){
   if [[  $flag -eq 1 ]]; then
         echo "Error!You can't use option 'n' with option 't','a' ou
'i'"
           exit 1
       fi
}
```

Por fim nesta secção de código, encontra-se a função ***verifyflag(),*** cujo uso é muito importante na validação das ordenações. Como irá ser visto no Processamento das Opções, só podem ser utilizadas uma das opções -n,-t,-a ou -i ,portanto esta função impede que o programa seja “corrido” caso que a variável global booleana ***flag*** tome o valor 1, imprimindo uma mensagem de erro e terminando o programa, tal como a verificação efetuada na função ***processdata()***.

Processamento De Opções

```
#-----------Processamento de Opçoes-------------------
while getopts 'e:g:s:u:f:rntai' OPTION; do
 case "$OPTION" in
   g)
       g="$OPTARG"
       if  ! getent group $g >/dev/null 2>&1 ; then
             echo "Not possible to perform option '-g'  because
there's no group associated with $g"
             exit 1

       else
       groupname=$(getent group $g |awk -F: '{print $1}') #get name of
the group of $optarg
       usersarray=($(last -f ${file}| grep -v 'reboot\|wtmp' | awk
'{print $1}' | sort | uniq)) #get all users
       arrayusersgroup=""
           for i in ${usersarray[@]}
             do
               if  id -ng $i >/dev/null 2>&1; then
                   groupuser=$(id -ng $i)         #check the group of
each user , probably better ways to do than a for loop...
               if [[ $groupuser == $groupname && ${#arrayusersgroup}
!= 0 ]] ; then
                 arrayusersgroup+="\|" #regex to be able to do "grep"
to multiple words at a time,however we dont want it on the last index,
thats the reason of the if condition
                 arrayusersgroup+=$i
               elif [[ $groupuser == $groupname && ${#arrayusersgroup}
== 0 ]] ; then
                 arrayusersgroup+=$i
               fi
               fi
             done
       a="${a} | grep '${arrayusersgroup}'"
       fi
       ;;
```

Dentro deste fragmento de código, utilizamos as funcionalidades da função ***getopts,***para conseguir processar e tratar das opções escolhidas pelo utilizador. Nesta

temos como opções válidas as que estão contidas em `'e:g:s:u:f:rntai'`, onde aquelas que irão necessitar de um argumento respetivo têm, nesta expressão, o carácter **":"** , enquanto que as que não,`'rntai'`, não o possuem.

Como primeira opção válida, tem-se a opção ***-g***, responsável por mostrar os utilizadores dentro do grupo associado ao parâmetro passado após a opção -g. Para concretizar esta filtragem, como ponto de partida, confirma-se se existe grupo associado ao argumento passado, ***OPTARG***: caso não exista, alerta-se o utilizador do programa de tal e terminamos este. Seguidamente, guardamos mais uma vez todos os utilizadores num array temporário ,e, através dum ciclo tentamos encontrar correspondência entre o grupo de cada um destes, e o de ***OPTARG*** , caso haja sucesso o nome do utilizador e concatenado a uma string que os separa por "**\|*".*** A razão desta separação tem a ver com a forma como iremos filtrar o conteúdo da variável ***a***: pesquisar por múltiplas palavras através de um novo uso de ***grep***.

```
s)
       if ! date -d "$OPTARG" "+%Y-%m-%d" >/dev/null 2>&1; then
         echo "Please insert a correct date format"
             exit 1;
       fi
       horas=${OPTARG:6}
       if [[ $horas == *":"* ]]; then

           total=$(date -d "$OPTARG" +"%Y-%m-%d%H:%M")
           a="${a} -s $total"
       else

           total=$(date -d "$OPTARG" +"%Y-%m-%d")
           a="${a} -s $total"
       fi
       ;;
     e)
         if ! date -d "$OPTARG" "+%Y-%m-%d " >/dev/null 2>&1; then
         echo "Please insert a correct date format"
             exit 1;
       fi
       horas=${OPTARG:6}
       if [[ $horas == *":"* ]]; then
           total=$(date -d "$OPTARG" +"%Y-%m-%d%H:%M")
```

```
            a="${a} -t $total"
        else
            total=$(date -d "$OPTARG" +"%Y-%m-%d")
            a="${a} -t $total"
        fi

        ;;
```

Caso se pretenda realizar a filtragem de resultados por um intervalo de tempo particular, disponibiliza -se as opções ***-s*** e -e, cujo objetivo consiste em mostrar resultados após uma data específica, e até outra data (também intrínseca), respetivamente. Estas duas opções são bastante idênticas, diferindo no uso das propriedades do comando last, algo que irá ser demonstrado agora.

Cada uma destas opções tem como base uma verificação bastante concisa da data introduzida impedindo a introdução de dias superiores a 31, caso haja engano na introdução do nome do mês respetivo, horas superiores a 24 ou inferiores a zero,etc.

A variável ***horas***, contém o carácter na posição 6 do ***OPTARG***, ":", tem o intuito de verificar quando o utilizador introduziu horas ou não. Isto foi necessário pois sem `%H:%M,` a string ***total*** , que contém o resultado do comando date ,neste script desenvolvido, não leria as horas no processamento dos dados, ocorrendo um erro.

Assim, a variável ***a*** é concatenada com "***-s total"***,no caso da opção -***s*** , pois ***"last -s 'time' "*** filtra o resultado do comando tendo como base as sessões a partir dessa data. Pelo contrário na opção -***e,*** concatena-se como ***"-t total"***.

```
  u)
        # se contiver apenas numeros nao e valido, utilizadores não
podem ter como nome apenas numeros
        if [[ -n ${OPTARG//[0-9]} ]] ; then
          u="$OPTARG"
          a="${a} | grep '$u'"
        else
          echo "Invalid! Make sure you didn't introduce only numbers!"
          exit 1
        fi
        ;;
   f)
        if [[ ! -f $OPTARG ]]; then
          echo "The file introduced  does not not exist!"
          exit 1
        else
          f="$OPTARG"
          file=${f}
```

```
        a="${a} -f ${file}"
      fi
      ;;
```

Efetuando a opção ***-u*** processam-se os dados da variável a ,consoante uma expressão regular, ou um padrão encontrado. Como trabalho de pesquisa e curiosidade, conclui-se que os *usernames* em *Linux* não podem ser constituído por apenas números, portanto caso tente-se introduzir apenas números invés de uma expressão regular, irá ser apresentada uma mensagem de erro, e consequentemente fechar o programa.

```
  f)
      if [[ ! -f $OPTARG ]]; then
        echo "The file introduced  does not not exist!"
        exit 1
      else
        f="$OPTARG"
        file=${f}
        a="${a} -f ${file}"
      fi
      ;;
```

Como o comando ***last*** permite o *display* das informações a partir de outros ficheiros diferentes do predefinidos, é pertinente haver uma opção que permita essa funcionalidade.Neste aspeto, encontra-se neste fragmento de código, a opção -f que possibilita o referido ainda agora.

A implementação baseia-se em novamente verificar se o argumento introduzido é válido, ou seja, se ***OPTARG***,guardado na variável ***f***, é de facto um ficheiro existente no sistema computacional.Caso ocorra um sucesso junta-se a variável ***a*** com "-f ${file}", e em sentido adverso, ocorre um erro pelo que se avisa o user de tal e termina-se o programa.

```
        r)
            varsort=" ${varsort} -r"
            ;;
```

```
        n)
            verifyflag
                    varsort=" ${varsort} -n -k2"
            flag=1
            ;;
        t)
            verifyflag
            varsort=" ${varsort} -n -k3"
            flag=1
            ;;
        a)
            verifyflag
            varsort=" ${varsort} -n -k4"
            flag=1
            ;;
        i)
            verifyflag
            varsort="  ${varsort} -n -k5"
            flag=1
            ;;
        \?)
             usage
             exit 0
          ;;
        :)
          echo "Option -$OPTARG requires an argument to be executed!." >&2
          exit 1
          ;;
      esac
      done
      shift "$(($OPTIND -1))" #the $OPTIND variable is set to 1, and it is
incremented each time an option is parsed, until it reaches the last one.
      processdata
      printf "Tamanho do array: %s\n" ${#userstatsarray[@]}
      printf  "%-8s\n" "${userstatsarray[@]}" | ${varsort}
```

As últimas opções que são tratadas, correspondem à ordenação dos dados que se pretende efetuar:

- ***-r***, para ordenação decrescente;
- ***-n***, para ordenar pela segunda coluna, ou seja, o número de sessões (sort -n -k2);
- ***-t***, com vista à ordenação pela terceira coluna, tempo total das sessões por utilizador (sort -n-k3);

- ***-a***, em que se ordena pela quarta coluna, em que se ordena por tempo máximo (sort -n -k4);
- ***-i***, em que se ordena pela quinta e última coluna, tempo mínimo (sort -n -k4);

Em todas estas opções chamamos a função de verificação de *inputs*, ***verifyflag()***, com a exceção clara da opção -r, pois podemos ordenar de forma reversa qualquer das colunas.Para que esta mesma função tenha o desempenho que se pretende a variável *flag* passa a ter valor 1.

Terminando, se a opção não seja reconhecida esta entra no case "***-?)***", e chama-se a função ***usage()***, se, por outro lado, não for introduzido um argumento para as opções que requerem isso, ativa-se o case "***:)***" fazendo display da respetiva mensagem de erro e claramente a saída do programa.

A linha ***shift $((OPTIND-1))*** remove todas as opções da lista de argumentos que já foram analisadas pelo ***getopts,*** depois disto, ***$1*** referir-se-á ao primeiro argumento passado ao programa que não é uma opção de ordenação.

Main

```
processdata
#printf "Tamanho do array: %s\n" ${#userstatsarray[@]}
printf  "%-8s\n" "${userstatsarray[@]}" | ${varsort}
```

Este script apresenta como ***Main*** do programas apenas 2 linhas de código.A primeira linha chama a função que podemos denominar de principal visto que é onde ocorre tudo, e, por fim imprimimos o *array* tendo sempre em conta o valor da variável ***varsort***, que, como foi visto , contém o tipo de ordenação escolhida ao executar o programa. A impressão do array tem, por cada elemento, a seguinte forma: <*username*> <nº sessões> <duração total> <tempo máximo> <tempo mínimo>.

Testes Efetuados

Para confirmar e concluir se todo o código do script se apresentava funcional para aquilo que se pretendia, foram realizados vários testes que irão ser apresentados agora.

Como o computador utilizado para os testes apresentados apresenta apenas um utilizador, decidiu-se testar recorrendo frequentemente a um ficheiro ***wtmp*** com as sessões e respetivos users, de um outro computador

```
danielgomes@danielgomes-Lenovo-ideapad-530S-14IKB:~/Documents$ ./userstats.sh -f ./wtmp
nlau 6 11 4 0
sd0104 4 579 230 50
sd0105 10 129 44 0
sd0106 1 0 0 0
sd0109 2 144 131 13
sd0301 60 0 0 0
sd0302 256 1399 133 0
sd0303 28 4 2 0
sd0304 1 0 0 0
sd0305 20 0 0 0
sd0401 21 0 0 0
sd0402 90 133 131 0
sd0403 1 9 9 9
sd0405 610 46 10 0
sd0406 182 354 154 0
sd0407 2 186 136 50
sop0101 14 1614 1314 0
sop0106 1 0 0 0
sop0202 17 2607 1530 3
sop0301 3 63191 31526 139
sop0402 18 2851 1439 35
sop0406 10 305 165 0
```

O resultado utilizando a opção ***-f*** foi o esperado, onde encontra-se os resultados ordenados por ordem alfabética da primeira coluna.

Utilizando de forma conjunta a opção *-f* com a opção ***-u***, obteve-se o seguinte:

```
danielgomes@danielgomes-Lenovo-ideapad-530S-14IKB:~/Documents$ ./userstats.sh -f ./wtmp -u "sop.*"
sop0101 14 1614 1314 0
sop0106 1 0 0 0
sop0202 17 2607 1530 3
sop0301 3 63191 31526 139
sop0402 18 2851 1439 35
sop0406 10 305 165 0
```

Recorrendo à opção ***-g*** apresenta-se nesta imagem, o sucedido quando introduz-se um grupo de *users* não válido , e também quando é válido:

```
danielgomes@danielgomes-Lenovo-ideapad-530S-14IKB:~/Documents$ ./userstats.sh -g sop
Not possible to perform option '-g'  because there's no group associated with sop
danielgomes@danielgomes-Lenovo-ideapad-530S-14IKB:~/Documents$ ./userstats.sh -g danielgomes
danielgo 42 64594 16556 0
```

Para as opções que filtram por tempo:

```
danielgomes@danielgomes-Lenovo-ideapad-530S-14IKB:~/Documents$ ./userstats.sh -f ./wtmp -s "Jun 5 17:30" -e "Jun 7 00:00"
sd0104 8 1158 230 50
sd0109 2 26 13 13
sd0301 2 0 0 0
sd0302 22 276 132 0
sd0303 20 4 2 0
sd0304 2 0 0 0
sd0401 42 0 0 0
sd0402 38 0 0 0
sd0405 18 0 0 0
sd0406 68 18 4 0
sd0407 4 372 136 16
```

```
danielgomes@danielgomes-Lenovo-ideapad-530S-14IKB:~/Documents$ ./userstats.sh -f ./wtmp -s  "Jun 5 25:30" -e "Jun 7 00:00"
Please insert a correct date format
danielgomes@danielgomes-Lenovo-ideapad-530S-14IKB:~/Documents$ ./userstats.sh -f ./wtmp -s  "Jun 5 25:30" -e "lasddsad 7 00:00"
Please insert a correct date format
danielgomes@danielgomes-Lenovo-ideapad-530S-14IKB:~/Documents$ ./userstats.sh -f ./wtmp -s  "Jun 5 17:30" -e "lasddsad 7 00:00"
Please insert a correct date format
```

Finalmente, testando as opções por ordenação:

```
danielgomes@danielgomes-Lenovo-ideapad-530S-14IKB:~/Documents$ ./userstats.sh -f ./wtmp -t -r -u "sop.*"
sop0301 3 63191 31526 139
sop0402 18 2851 1439 35
sop0202 17 2607 1530 3
sop0101 14 1614 1314 0
sop0406 10 305 165 0
sop0106 1 0 0 0
danielgomes@danielgomes-Lenovo-ideapad-530S-14IKB:~/Documents$ ./userstats.sh -f ./wtmp -n -r -u "sop.*"
sop0402 18 2851 1439 35
sop0202 17 2607 1530 3
sop0101 14 1614 1314 0
sop0406 10 305 165 0
sop0301 3 63191 31526 139
sop0106 1 0 0 0
danielgomes@danielgomes-Lenovo-ideapad-530S-14IKB:~/Documents$ ./userstats.sh -f ./wtmp -a -r -u "sop.*"
sop0301 3 63191 31526 139
sop0202 17 2607 1530 3
sop0402 18 2851 1439 35
sop0101 14 1614 1314 0
sop0406 10 305 165 0
sop0106 1 0 0 0
danielgomes@danielgomes-Lenovo-ideapad-530S-14IKB:~/Documents$ ./userstats.sh -f ./wtmp -i -r -u "sop.*"
sop0301 3 63191 31526 139
sop0402 18 2851 1439 35
sop0202 17 2607 1530 3
sop0406 10 305 165 0
sop0106 1 0 0 0
sop0101 14 1614 1314 0
danielgomes@danielgomes-Lenovo-ideapad-530S-14IKB:~/Documents$ ./userstats.sh -f ./wtmp -t -a -r -u "sop.*"
Error!You can't use option 'n' with option 't','a' ou 'i'
danielgomes@danielgomes-Lenovo-ideapad-530S-14IKB:~/Documents$
```

Terminal

Comparestats.sh

Script responsável por comparar dois ficheiros consequentes do script userstats.sh e por retornar a diferença entre os tempos de utilização e número de sessões. Este programa recebe o caminhos para os dois ficheiros como argumentos, sendo o primeiro argumento referente aos valores mais recentes. Tal como o *userstats*.sh, os dados também podem ser ordenados recorrendo às opções ***-r***, ***-t***, ***-n***, ***-a***, ***-i***.

Variáveis Globais

```
#-----Variaveis globais-----
fileA=""
fileB=""
a=()
b=()
tempFinal=()
final=()
ordenador="sort"
flag=0;
```

Nesta fração de código são definidas todas as variáveis usadas em vários pontos do código, de onde se retiram as variáveis ***fileA*** e ***fileB*** responsáveis por armazenar os nomes dos ficheiros - passados por via de argumentos. A variável ***ordenador*** ficará com um excerto de código bash correspondente à ordenação; o seu valor inicial é ***'sort'*** uma vez que serão adicionados à variável o(s) filtros de ordenação. Os vetores ***a()*** e ***b()*** recolherá as linhas - registros - presentes nos ficheiros lidos. O ***tempFinal()*** é apenas um meio para armazenar dados temporariamente antes de serem filtrados para a variável ***final().***

Funções

```
#--------funçoes-----------
function usage (){
   echo "Options Error! "
   echo ' OPTION                    Description                    '
   echo ' Sorting : '
   echo "          -r:              Sort all data in reverse order"
   echo "          You can not use more than 1 option from these:"
   echo ' '
   echo "          -n:              Sort by number of sessions"
   echo "          -t:              Sort by total logged time"
   echo "          -a:              Sort by maximum logged time"
   echo "          -i:              Sort by minimum logged time"
}
function verifyflag(){
   if [[  $flag -eq 1 ]]; then
            echo "Error!You can't use option 'n' with option 't','a' ou 'i'"
            exit 1
       fi
}
mfcalculations () {
   indiceInicial=$1
   registoA=(${a[$indiceInicial]}) # Daniel 56 0 0 0
   if [[ ${#a[@]} -eq $indiceInicial ]]; then
       return 1
   fi
   if [[ " ${b[*]} " == *" ${registoA[0]} "* ]]; then
       for ((i=0;i<${#b[@]};i++)); do
           registoB=(${b[$i]}) # Andre 15 0 0 0
           if [[ ${registoA[0]} == ${registoB[0]} ]]; then
               nome=${registoA[0]}
               nsessoes=$((${registoA[1]} - ${registoB[1]}))
               dtotal=$((${registoA[2]} - ${registoB[2]}))
               dmaxima=$((${registoA[3]} - ${registoB[3]}))
               dminima=$((${registoA[4]} - ${registoB[4]}))
               tempFinal+="$nome $nsessoes $dtotal $dmaxima $dminima "
           elif [[ ! s" ${a[*]} " == *" ${registoB[0]} "* ]] && [[ ! s"
${tempFinal[*]} " == *" ${registoB[0]} "* ]]; then
               tempFinal+="${registoB[@]} "
           fi
       done
   else
```

```
        tempFinal+="${registoA[@]} "
    fi
    mfcalculations $[$indiceInicial+1]
}
```

A sequência de código acima define três funções: ***usage(), verifyflag()*** e ***mfcalculations()***.

A primeira é chamada quando o utilizador não invoca o script corretamente, indicando quais os argumentos de ordenação corretos. Neste script são apenas permitidas as opções -r, -n, -t, -a, -i, sendo que as últimas quatro não podem ser usadas em simultâneo, uma vez que se referem à ordenação de um coluna em específico - não se pode ordenar os registos por ordem crescente da segunda coluna e por ordem crescente da terceira coluna em simultâneo.

A segunda função, ***verifyflag()*** é uma função de controlo que verifica se já foi utilizada alguma das quatro opções de ordenação únicas. No caso de a variável flag ter um valor superior a 0, isto é, o valor 1, então significará que foram introduzidas pelo menos duas opções de ordenação e o programa terminará com uma saída de erro.

A terceira função, ***mfcalculations()*** é uma função recursiva que recebe como argumento o índice correspondente ao registo do ficheiro A (o ficheiro mais recente) que será analisado. No caso de o índice não pertencer ao array então a função retorna um valor arbitrário, neste caso 1, de maneira a continuar as instruções presentes mais abaixo. De seguida é feita uma verificação para ver se há coincidências entre o registo presente (***registoA***) e os registos do ficheiro B - registos mais antigos - e no caso de não existir coincidências então, à variável tempFinal, é adicionado o ***registoA*** sem que hajam quaisquer calculos. Se o utilizador do ***registoA*** estiver nos registos do ficheiro B, então procede-se à procura* pelo registo cujo nome de utilizador coincide com o do ***registoA*** e procede-se aos cálculos: número de sessões, duração total, duração máxima e duração mímina. Todos estes dados são compilados na variável tempFinal para posterior processamento. Ainda durante esta procura* no caso de o ***registoB*** - registo do ficheiro B - não ter quaisquer semelhanças com os registos do ficheiro A, então todo o registo (nome, duração total, número de sessões, duração máxima e duração mínima) é adicionado à variável ***tempFinal***. Como a função é recursiva, então, no final, é incrementado um valor ao índice para que se possa calcular os tempos/sessões para o registo seguinte.

Processamento de Opções

```
#---------Tratamento de Opçoes-----------------
while getopts 'rntai' OPTION;do
   case "$OPTION" in
       r)
           ordenador="${ordenador} -r"
           ;;
       n)
           verifyflag
           ordenador="${ordenador} -n -k2"
           flag=1
           ;;
       t)
           verifyflag
           ordenador="${ordenador} -n -k3"
           flag=1
           ;;
       a)
           verifyflag
           flag=1;
           ordenador="${ordenador} -n -k4"
           ;;
       i)
           verifyflag
           ordenador="${ordenador} -n -k5"
           flag=1
           ;;
       \?)
        usage
        exit 0
     ;;
   esac
done
shift "$(($OPTIND -1))"
```

Neste fragmento de código é utilizada a função ***getopts*** - bastante útil para fazer a especificação dos argumentos introduzidos pelo utilizador: -r, -n, -t, -a, -i. No caso das últimas 4 opções, além de escrita na variável ***ordenador*** a opção de ordenação é ainda verificada se já existe alguma das outras opções e fixa a variável ***flag*** a 1.

Main

```
#----------Main----------------
if [[ ! $# -eq 2 ]]; then
   echo ""
   echo "#--------------- Invalid number of arguments ---------------#"
   echo ""
   exit
fi

if [[ $# -eq 2 ]]; then
   if [[ ! -f $1 || ! -f $2 ]]; then
       echo "No file found!"
       exit
   fi
   fileA=$1
   fileB=$2
fi
while IFS= read line
do
   a+=("$line")
done <"$fileA"

while IFS= read line
do
   b+=("$line")
done <"$fileB"
```

É neste ínicio da secção ‘Main’ que se procede à leitura dos ficheiros mas não sem antes verificar se o número de argumentos é igual a 2. Constatando que o programa é chamado com um número diferente de 2 argumentos, então o programa termina com uma mensagem de erro, indicando que o número de argumentos está incorreto. Seguidamente, e partindo do partindo do princípio de que há 2 e apenas 2 argumentos (os dois ficheiros), é feita a verificação se os ficheiros foram encontrados, senão, o programa acaba com uma mensagem de erro.

```
while IFS= read line
do
   a+=("$line")
done <"$fileA"
```

```
while IFS= read line
do
   b+=("$line")
done <"$fileB"

mfcalculations 0
els=($(echo $tempFinal | tr " " "\n"))

for ((i=0;i<${#els[@]};i++)); do
   if (( $i % 5 == 0 )); then
       final+=("${els[$i]} ${els[$i+1]} ${els[$i+2]} ${els[$i+3]} ${els[$i+4]}")
   fi
done
   printf  "%s\n" "${final[@]}" | ${ordenador}
```

Esta segunda parte do código da secção 'Main' começa por ler os ficheiros e por guarda nos arrays ***a()*** e ***b()*** os registos correspondentes aos ficheiros mais recente e mais antigo, respetivamente. Feito isto, é chamada a função mfcalculations com o argumento 0, significando que começa a ler o primeiro registo do array ***a()*** e o array ***els()*** armazena todas as palavras (delimitadas por espaço) para que, percorrendo o todos os elementos deste vetor, se possa adicionar um novo elemento ao vetor final de maneira que esse elemento seja formatado da seguinte forma: <username> <nº sessões> <duração total> <tempo máximo> <tempo mínimo>. Na verdade, a variável ***tempFinal*** é uma String que armazena, separada por espaços, os valores. Analisando, é possível reparar que cada elemento tem 5 palavras, assim sendo, o resto da divisão ser zero indica-nos que este será o <username> do registo que será adicionado ao array ***final***.

Testes Efetuados

Com o objetivos de confirmar de to script estaria funcional, foram realizados os seguintes testes. Foi necessário utilizar uma workstation da universidade como forma de confirmar o funcionamento do programa, uma vez que os computadores pessoais de ambos os elementos do grupo têm apenas um utilizador.

ps: os ficheiros de texto usados no teste encontram-se na Workstation 04 - grupo 06

```
[sop0406@l040101-ws04 ~]$ ./userstats.sh -n ".*" > userstats_20191012
[sop0406@l040101-ws04 ~]$ ./userstats.sh -s "Nov 11 11:11" > userstats_20191112
[sop0406@l040101-ws04 ~]$
```

Os ficheiros começam por ser inicializados com os dados retornados pelo script ***userstats.sh*** mas com argumentos de filtragem diferentes.

```
[sop0406@l040101-ws04 ~]$ ./comparestats.sh userstats_20191111 userstats_20191011
sop0101 1 339 339 39
sop0102 0 0 0 0
sop0202 1 0 0 0
sop0404 14 1023 343 0
sop0405 4 335 132 5
sop0406 -2 -74 0 0
sop0407 6 435 238 4
sop0409 1 2 2 2
```

Retorno sem argumentos.

```
[sop0406@l040101-ws04 ~]$ ./comparestats.sh -a -i userstats_20191111 userstats_20191011
Error!You can't use option 'n' with option 't','a' ou 'i'
```

Retorna erro se forem introduzidos dois argumentos únicos.

```
[sop0406@l040101-ws04 ~]$ ./comparestats.sh -a -r userstats_20191111 userstats_20191011
sop0404 14 1023 343 0
sop0101 1 339 339 39
sop0407 6 435 238 4
sop0405 4 335 132 5
sop0409 1 2 2 2
sop0406 -2 -74 0 0
sop0202 1 0 0 0
sop0102 0 0 0 0
```

Retorno do script com dois argumentos: ***-a*** e ***-r*** que ordena o resultado por ordem inversa pela quarta coluna.

```
[[sop0406@l040101-ws04 ~]$ ./comparestats.sh -t userstats_20191111 userstats_20191011
sop0406 -2 -74 0 0
sop0102 0 0 0 0
sop0202 1 0 0 0
sop0409 1 2 2 2
sop0405 4 335 132 5
sop0101 1 339 339 39
sop0407 6 435 238 4
sop0404 14 1023 343 0
```

Números ordenados pela coluna do tempo total.

Segundo os resultados dos teste é possível verificar que o script está funcional quando utilizado em sistemas operativos *Linux*. Quando testado no MacOS 10.14.6 ocorria um erro relacionado com a invalidade da data.

Conclusão

Com este trabalho pretendemos aprofundar os nossos conhecimentos da linguagem de programação da Shell, criando dois scripts que, mutuamente, criam estatísticas de utilizadores, recorrendo ao ficheiro ***wtmp*** presente no computador. Assim, concluímos que há formas mais eficientes de se manipular dados e que a shell, embora simples de se utilizar, torna-se mais complexa quando utilizada na formatação e manipulação de grandes quantidades de dados.