

Sistemas Operativos

Professor:
José Nuno Panelas Nunes Lau

Estatísticas de Utilizadores em Bash

Carolina Araújo, 93248
Hugo Paiva, 93195

Igual distribuição de trabalho
entre os dois membros



DETI
Universidade de Aveiro
29-11-2019

Índice

1	Introdução	2
2	Preparação	3
3	Estatísticas dos utilizadores	4
3.1	Estrutura	4
3.2	Tratamento de Opções	5
3.3	Leitura e tratamento de dados	7
3.4	Impressão ordenada dos dados filtrados	11
4	Comparação das estatísticas dos utilizadores	12
4.1	Estrutura	12
4.2	Tratamento de Opções	13
4.3	Leitura e tratamento de dados	15
4.4	Impressão ordenada dos dados filtrados	17
5	Resultados	19
5.1	Estatísticas dos utilizadores	20
5.2	Comparação das estatísticas dos utilizadores	26
6	Conclusão	29
7	Bibliografia	30

1 Introdução

Este trabalho prático foi baseado no desenvolvimento de scripts em Bash que permitem recolher algumas estatísticas sobre o modo como os utilizadores estão a usar o sistema computacional.

Estas ferramentas permitem visualizar o número de sessões e o tempo total de ligação para uma selecção de utilizadores e um determinado período de tempo, permitindo também a comparação dos dados obtidos em períodos distintos.

Para desenvolver estas ferramentas com os resultados expectáveis é necessário compreender o funcionamento da Bash. Sendo mantida pelo famoso projeto GNU, a Bash, é uma ferramenta extremamente eficiente encontrada na maioria dos sistemas baseados em UNIX. É também altamente personalizável e, por isso, muito usada no mundo da programação.

2 Preparação

Antes de avançar com qualquer desenvolvimento em código, procedeu-se à cópia do ficheiro `"/var/log/wtmp"` do computador da sala de aula. Este ficheiro contém o histórico atual de todos os logins e logouts associados ao computador em questão, fornecendo todos os dados necessários para o desenvolvimento dos scripts. Desta forma, foi permitida a sua leitura, durante a implementação, sem necessidade de se estar conectado ao computador nas instalações do DETI.

Um dos membros do grupo, devido a possuir um Macbook, apenas dispunha da versão 3 da Bash. Sendo esta versão de 2007, várias funcionalidades que viriam a ser utilizadas durante o trabalho não eram suportadas. Um exemplo disto são os *Arrays Associativos* que apenas são suportados a partir da versão 4 da Bash, versão esta que está instalada nos computadores do DETI. Este problema foi resolvido através da utilização de uma máquina virtual de ambiente Linux.

3 Estatísticas dos utilizadores

O script criado (*userstats.sh*) permite a visualização filtrada das estatísticas dos utilizadores contidas no ficheiro `/var/log/wtmp`. Após a sua execução, é possível observar o número de sessões, tempo total de ligação (em minutos), duração máxima e duração mínima das sessões dos utilizadores seleccionados, no período desejado.

3.1 Estrutura

Inicialmente, o grupo começou o trabalho utilizando apenas *Arrays* para armazenar e tratar os dados. Após a descoberta dos *Arrays Associativos* e a facilidade que estes dispõem em associar, neste caso, os utilizadores com a informação dos mesmos, foi estabelecida a estrutura final onde os dados iriam ser tratados.

Posto isto, foram criados dois *Arrays Associativos* e um *Array* para simplificar o tratamento dos dados.

```
3 # Arrays para guardar users e a sua informação
4 users=() #Array onde são guardados os utilizadores únicos
5 declare -A argOpt=() #Array Associativo onde são guardadas as opções passadas
6 declare -A userInfo=() #Array Associativo onde é guardada a informação após o tratamento de dados correspondente a cada utilizador
```

Figure 1: *Arrays* mais relevantes na implementação deste script

- **users** - *Array* onde são guardados os utilizadores, sem repetição, durante o tratamento de dados;
- **argOpt** - *Array Associativo* onde são guardadas as opções e os respetivos argumentos durante o tratamentos de opções;
- **userInfo** - *Array Associativo* onde são guardadas as informações tratadas durante o tratamento de dados.

Ao armazenar a informação desta forma, principalmente devido aos *Arrays Associativos*, o tratamento de dados, tal como a impressão dos mesmo são muito mais facilitada.

3.2 Tratamento de Opções

As opções foram tratadas usando o comando shell *getopts*, abordado nas aulas práticas. Este comando analisa os argumentos passados na Bash e, se estes estiverem definidos na sua criação, são efetuadas as tarefas desejadas.

O tratamento de opções é iniciado com a chamada da função *args()*.

```
22 # Tratamento de opções
23 function args() {
24
25     repeat=0;
26     while getopts g:u:s:e:f:rntai option; do
27         case "${option}" in
28             g) #Seleção de utilizadores através do seu grupo
29                 if [ ${OPTARG:0:1} == "-" ]; then
30                     usage
31                 fi
32                 ;;
33             u) #Seleção de utilizadores através do nome dos utilizadores
34                 if [ ${OPTARG:0:1} == "-" ]; then
35                     usage
36                 fi
37                 ;;
38             s) #Seleção do período através da especificação da data a partir da qual as sessões são consideradas
39                 if [ ${OPTARG:0:1} == "-" ]; then
40                     usage
41                 fi
42                 ;;
43             e) #Seleção do período através da especificação da data a partir da qual as sessões não são consideradas
44                 if [ ${OPTARG:0:1} == "-" ]; then
45                     usage
46                 fi
47                 ;;
48             f) #Extração das informações a partir de um ficheiro distinto
49                 if [ ${OPTARG:0:1} == "-" ]; then
50                     usage
51                 fi
52                 ;;
53             r);;
54             n | t | a | i)
55                 if [[ $repeat = 1 ]];then
56                     usage
57                 else
58                     repeat=1
59                 fi
60                 ;;
61             *) #Opções não atribuídas pelo getopts
62                 usage
63                 ;;
64             esac
65
66             if [[ -z "$OPTARG" ]]; then
67                 argOpt[$option]="none"
68             else
69                 argOpt[$option]="${OPTARG}"
```

Figure 2: Função *args()* que procede ao tratamento de opções

O *getopts* recebe parâmetros (*g:u:s:e:f:rntai*) que correspondem às opções que o comando aceita. As opções *-g*, *-u*, *-s*, *-e* e *-f* recebem argumentos pois estão seguidas de dois pontos ':'. Estando este comando dentro de um *while*, é repetido as vezes necessárias para percorrer todas as opções e argumentos introduzidos no terminal, aquando a chamada do programa. As opções são guardadas, em cada ciclo, na variável *option* sendo executado o código associado a cada opção, consoante a *case statement*.

Nas opções válidas que recebem argumentos foi feita uma verificação se o argumento introduzido não é uma outra opção (por erro do utilizador). Isto é feito comparando o 1º caractere do argumento com '-', executando a

função *usage* quando esta comparação é verificada, significando que é uma outra opção.

Para as opções *-n*, *-t*, *-a* e *-i* que não podem ser repetidas, foi criada uma variável *repeat* para averiguar se alguma destas variáveis já foi passada à função. Em caso afirmativo, é executada a função *usage()*.

Sempre que é introduzida uma opção inválida ou de forma incorreta, o *case statement* executa a função *usage()*, que indica a forma de utilização do script.

```
8  # Usage do script
9  function usage() {
10     echo "Usage: $0 -g [grupo] -u [nome] -s [data1] -p [data2] -r -n -t -a -i"
11     echo ""
12     echo "[grupo] = Grupo de utilizadores"
13     echo "[nome] = Nome dos utilizadores"
14     echo "[data1] = Data de início da sessão a partir da qual as sessões devem ser consideradas"
15     echo "[data2] = Data de início de sessão a partir da qual as sessões não devem ser consideradas"
16     echo ""
17     echo "Todas estas opções são opcionais, sendo que o script corre sem nenhuma opção."
18     echo ""
19     exit
20 }
```

Figure 3: Função *usage()* que refere as opções e argumentos esperados

Ainda dentro do ciclo *while*, são executadas averiguações para guardar as opções e os respetivos argumentos numa das estruturas de dados definidas anteriormente. Se for passada uma opção válida mas nenhum argumento, ou seja, a variável *OPTARG* está vazia, a opção *-z* da expressão condicional retorna *true* e é guardado no *Array Associativo* *argOpt* a opção em questão (*key*) e o valor *"none"* (*value*). Ao passar uma opção válida com um argumento, é guardada na mesma estrutura de dados a opção em questão (*key*) e o valor do argumento (*value*).

```
66     if [[ -z "$OPTARG" ]]; then
67         argOpt[$option]="none"
68     else
69         argOpt[$option]=$OPTARG
70     fi
71 done
72
73     if [ ! $((OPTIND-1)) -eq $# ]; then
74         usage
75     fi
76
77     shift $((OPTIND - 1))
```

Figure 4: Verificação da inexistência de argumento associado à atual opção

Já fora do ciclo *while*, é executada a verificação se *\$((OPTIND-1))* é igual ao número de argumentos passado à função. Visto que a variável *OPTIND* corresponde às opções e argumentos aceites no *getopts*, incluindo o nome do ficheiro, e *\$#* corresponde ao número de argumentos passados ao script, com exceção do nome do ficheiro, é **subtraído um** ao *OPTIND* de modo a comparar o que foi aceite no *getopts* e o que não, executando a função *usage* no caso de argumentos a mais.

Por fim, *\$((OPTIND-1))* vai remover todas as opções que foram passadas pelo *getopts*, nos seus parâmetros. Desta forma, *\$1* vai referir o primeiro argumento passado ao script, que não é uma opção.

3.3 Leitura e tratamento de dados

O tratamento de dados é iniciado com a chamada da função *getUsers()*.

```
81 # Tratamento e leitura de dados
82 function getUsers() {
83     if [[ -v argOpt[f] ]]; then # -v em declarative arrays vai verificar se o elemento a seguir está no array
84         users=$(last -f "${argOpt['f']}" | awk '{if($10 !~ /in/) {print $1}}' | sort | uniq | sed '/reboot/d' | sed "/${argOpt['f']}/d")
85     else
86         # Filtrar users
87         if [[ -v argOpt[u] ]]; then
88             match="${argOpt['u']}"
89             users=$(last | awk '{if($10 !~ /in/) {print $1}}' | sort | uniq | sed '/reboot/d' | sed '/wtmp/d' | grep "$match")
90         elif [[ -v argOpt[g] ]]; then
91             group="${argOpt['g']}"
92             users=$(last | awk '{if($10 !~ /in/) {print $1}}' | sort | uniq | sed '/reboot/d' | sed '/wtmp/d')
93             index=0
94             for u in ${users[@]}; do
95                 userGroups=( $(id -G -n $u) )
96                 if ! [[ ${userGroups[@]} =~ $group ]]; then
97                     unset users[index]
98                     index=$((index + 1))
99                 fi
100             done
101         elif [[ -v argOpt[s] || -v argOpt[e] ]]; then
102             start=$(date -d "${argOpt['s']}" +"%Y-%m-%d ")
103             start+=$(echo "${argOpt['s']}" | awk '{print $3}')
104             start="-s \"${start}\" "
105             end=$(date -d "${argOpt['e']}" +"%Y-%m-%d ")
106             end+=$(echo "${argOpt['e']}" | awk '{print $3}')
107             end="-t \"${end}\" "
108             users=$(eval last $start $end | awk '{if($10 !~ /in/) {print $1}}' | sort | uniq | sed '/reboot/d' | sed '/wtmp/d')
109         else
110             users=$(last | awk '{if($10 !~ /in/) {print $1}}' | sort | uniq | sed '/reboot/d' | sed '/wtmp/d')
111         fi
112     fi
113 }
114 }
```

Figure 5: Função *getUsers()* que seleciona os utilizadores que são guardados no *Array users*

De modo a aumentar a eficiência e, desta forma, diminuir o tempo de execução do programa, foi decidido filtrar os utilizadores pretendidos nesta função, guardando-os num *Array*. Assim, qualquer próxima função precisa apenas de iterar sobre os utilizadores, já previamente selecionados, para obter informação relativa aos mesmos.

Consoante os argumentos passados no terminal e através de comandos Linux como *grep*, *awk*, *sed*, *sort* e *uniq* foi-nos possível filtrar somente utilizadores únicos que correspondessem a uma determinada expressão *RegEx*, assim como apenas aqueles de um ficheiro específico, os de um certo grupo ou utilizadores que tivessem iniciado sessão entre certas datas.

A lógica geral de funcionamento desta função passa por chamar apenas uma vez o comando *last*, com base nas opções passadas no terminal, de modo a obter o seu *output* e então selecionar apenas a informação que nos é conveniente.

Através do comando *awk* é possível obter somente os elementos na posição *\$1* caso estes não correspondam a um utilizador ainda conectado ao computador em questão. Essa verificação faz-se com um *if* que testa se os elementos na posição *\$10* correspondem à palavra "in".

Os comandos *sort* e *uniq* garantem a salvaguarda apenas dos utilizadores únicos. São, ainda, capazes de retirar aqueles cujo elemento na posição *\$1* corresponda a "reboot" ou ao nome do ficheiro (passado no terminal ou então, por default, o ficheiro *"/var/log/wtmp"*), com a chamada do comando *sed*.

Se é passada a opção "-f" no terminal, o comando *last* é chamado juntamente com "-f <filename>". Enquanto que, se forem passadas as opções "-s" e "-e", procede-se à transformação das strings passadas como datas, de modo

a ficarem no formato **YYYY-MM-DD hh:mm** e, posteriormente, o *last* será chamado com as opções "-s" e "-t" (*last* -s <date1> -t <date2>), utilizando *eval*, uma vez que este comando serve para construir um outro comando, concatenando argumentos.

No caso da opção "-u" ter sido passada no terminal, utiliza-se o comando *grep "\$match"*, de forma a que somente utilizadores cuja string identificadora do seu nome corresponda à expressão passada no terminal, fiquem armazenados no *Array users*. Essa tal expressão terá sido guardada no elemento *\$match* (correspondente ao valor associado à opção "-u")

Nota: As aspas em torno da variável *\$match* na opção "-u" foram colocadas uma vez que, aquando da realização de testes finais, notou-se que a chamada do programa com as opções " **-n -u ".*"** " dava um output diferente do esperado, sendo que, com qualquer outra expressão RegEx, o programa imprimia o esperado. Após alguma pesquisa chegamos à conclusão de que seria necessário colocar estas aspas.

Por fim, caso seja indicada a opção "-g", atribui-se à variável *group* o valor passado como argumento desta opção, isto é, o grupo que se deseja filtrar. Da mesma maneira que, como nas outras opções, vai-se buscar todos os utilizadores únicos, iterando-os, para ver quais deles pertencem ao grupo especificado. Ver a que grupos pertence cada utilizador faz-se através do comando *id -G -n <utilizador>*, cuja opção "-G" torna mandatório imprimir apenas os grupos suplementares e "-n", como opção prévia, imprime o nome dos grupos e utilizadores em vez dos seus IDs. Posto isto, caso nenhum dos grupos do utilizador em questão seja igual ao grupo agora guardado na variável *group*, recorrendo ao comando *unset*, esse usuário será retirado do *Array users*. A variável *index* serve para aceder aos elementos indesejados, sendo que, após a retirada de um dos mesmos, esta variável é sempre incrementada.

Seguidamente a informação obtida passa para a função **getUserInfo()**.

```
144 function getUserInfo() {
145
146     for user in ${users[@]}; do
147         # OPÇÃO -f
148         if [[ -v argOpt[f] ]]; then
149             sessions=$(last -f "${argOpt[f]}" | grep -o $user | wc -l)
150             time=$(last -f "${argOpt[f]}" | grep $user | awk '{print $10}' | sed '/in/d' | sed 's/()(/g')
151
152         # OPÇÃO -s -e
153         elif [[ -v argOpt[s] || -v argOpt[e] ]]; then
154             sessions=$(eval last $start $end | grep -o $user | wc -l)
155             time=$(eval last $start $end | grep $user | awk '{print $10}' | sed '/in/d' | sed 's/()(/g')
156
157         # qualquer outra
158         else
159             sessions=$(last | grep -o $user | wc -l)
160             time=$(last | grep $user | awk '{print $10}' | sed '/in/d' | sed 's/()(/g')
161         fi
162
163         min=3000000
164         max=0
165         total=0
166
167         for t in $time; do
168             calculateTime "$t"
169         done
170
171         userInfo[$user]=$(printf "%-8s %-5s %-6s %-5s %-5s\n" "$user" "$sessions" "$total" "$max" "$min")
172     done
173     printIt
174 }
```

Figure 6: Função **getUserInfo()** que utiliza o *Array users* para obter a informação relativa a cada utilizador

Para cada utilizador filtrado na função anterior, esta vai contar o número de sessões e o tempo total, mínimo e máximo de ligação para cada sessão de cada utilizador. Isto é feito com auxílio aos comandos *last* | *grep*, que selecionam a informação do *last*, de modo a que o utilizador seja iterado pelo ciclo *for*, passando-o como argumento do *grep*, sendo ainda possível ir buscar informação a um ficheiro, caso seja passada a opção *-f* no terminal.

O comando *wc -l* conta o número de linhas onde aparece esse utilizador, revelando assim o número de sessões para cada um.

No caso do tempo de ligação de uma dada sessão, utiliza-se também o *grep* e o *awk* para ir buscar a informação na posição *\$10* (o tempo decorrido em sessão), utilizando o *sed* para remover os parênteses em torno do valor que realmente queremos, passando esta informação final para o *Array time*.

Caso tenham sido selecionadas as opções *-s* e *-e*, vamos buscar a informação formatada anteriormente para a data-início e a data-final, chamando novamente o comando *last* com recurso ao *eval*, obtendo assim apenas as sessões e tempos de ligação entre as datas em questão.

São definidos valores iniciais de tempo máximo, mínimo e total de ligação do utilizador em questão e cada valor do *Array* de tempos de cada ligação é passado à função **calculateTime()**. Esta acaba por retornar já estes valores calculados, usando os anteriormente definidos para realizar comparações e, por fim, toda esta informação é colocada dentro de um *Array Associativo*, cuja *key* é o utilizador e o *value* é já a informação final formatada numa string, com espaçamento e alinhamento definidos.

Função *calculateTime()*, chamada na função *getUsersInfo()*

```
114 function calculateTime() {
115     time=$1
116
117     # Calcular tempo em minutos
118     if (($#time >= 7)); then
119         minlogged=$(echo $time | tr '+' ':' | awk -F: '{ print ($1 * 1440) + ($2 * 60) + $3 }')
120     else
121         minlogged=$(echo $time | awk -F: '{ print ($1 * 60) + $2 }')
122     fi
123
124     # Calcular o tempo total
125     total=$((total + $minlogged))
126
127     # Calcular o tempo mínimo
128     if ((minlogged < min)); then
129         min=$minlogged
130     fi
131
132     # Calcular o tempo máximo
133     if ((minlogged > max)); then
134         max=$minlogged
135     fi
136 }
```

Figure 7: Função *calculateTime()* que calcula o tempo máximo, mínimo e total para dado utilizador, com base no Array *time*

Para facilitar o entendimento do código, deu-se o nome de *time* ao argumento passado à função, que é um dado elemento do Array de tempos calculado para cada utilizador.

O argumento *time* pode estar no formato **dd+hh:mm** ou apenas **hh:mm**. O primeiro contém a informação de que o tempo daquela sessão foi **dd** dias, **hh** horas e **mm** minutos, enquanto que o segundo não chega a 24h de ligação, indicando apenas o número de horas e minutos gastos. Com base nisto, qualquer tempo cujo *length* total seja superior a 7, significa que transporta a informação de um certo número de dias, portanto, substitui-se o '+' por ':' através do comando *tr*, de modo a que o *awk -F:* possa ir buscar a informação contida no elemento *time* e separa-la em cada ':' que encontrar. Assim, somos deixados com 3 argumentos (\$1, \$2 e \$3), indicando, cada um, respetivamente, o número de dias, horas e minutos dessa sessão. Com isto, basta multiplicar o número de dias, \$1, por 1440 (o número de minutos/dia), multiplicar \$2 por 60 (o número de minutos/hora) e somar ambos esses resultados com o valor de minutos, \$3, obtido pelo *awk*. Caso o *length* do elemento seja inferior a 7, significa que não indica número de dias, ou seja, não é necessário substituir nenhum sinal '+', visto que não há. De resto a informação é processada de modo bastante semelhante. O resultado obtido de qualquer um destes procedimentos é armazenado na variável *minlogged*.

De seguida, acrescenta-se à variável *total* de um dado *user*, o número de minutos calculado (*minlogged*). Compara-se também este valor aos tempos mínimos e máximos até então desse mesmo utilizador. Desta forma, verifica-se se o *minlogged* é inferior ao valor da variável *min* ou superior ao valor da variável *max*, podendo ser atualizado o valor de uma das duas variáveis, caso um dos casos se verifique.

Assim sendo, garante-se que todos os valores de tempos de cada sessão de um dado utilizador são comparados, podendo devolver a informação de qual é o tempo total de ligação, o tempo mínimo e o tempo máximo, como pretendido.

3.4 Impressão ordenada dos dados filtrados

A impressão ordenada dos dados filtrados é feita com a chamada da função ***printIt()***, que é chamada na função ***getUsers()***.

```
168 function printIt() {
169     if [[ -v argOpt[r] ]]; then
170         # ordem decrescente(nome user)
171         order="-rn"
172     else
173         order="-n"
174     fi
175
176     if [[ -v argOpt[n] ]]; then
177         # ordenar por numero de sessoes
178         printf "%s\n" "${userInfo[@]}" | sort -k2,2 ${order}
179
180     elif [[ -v argOpt[t] ]]; then
181         # por tempo total
182         printf "%s\n" "${userInfo[@]}" | sort -k3,3 ${order}
183
184     elif [[ -v argOpt[a] ]]; then
185         # por tempo máximo
186         printf "%s\n" "${userInfo[@]}" | sort -k4,4 ${order}
187
188     elif [[ -v argOpt[i] ]]; then
189         # por tempo mínimo
190         printf "%s\n" "${userInfo[@]}" | sort -k5,5 ${order}
191
192     else
193         #ordem crescente (nome user)
194         printf "%s\n" "${userInfo[@]}" | sort -k1,1 ${order}
195     fi
196 }
197
```

Figure 8: Função ***printIt()*** que procede à impressão dos dados tratados consoante a ordem desejada

O resultado ordenado dos dados filtrados é baseado na leitura e impressão, com a respetiva ordenação, dos dados dos *Arrays Associativos* ***argOpt*** e ***userInfo***.

No início da função, é verificada a existência da opção ***"-r"*** no ***argOpt*** através de uma expressão condicional com o operador ***-v***. Se isto acontecer, é criada uma variável chamada ***order*** onde é guardado o valor ***"-rn"***, valor esse que permite aos ***printf***'s imprimirem as informações ordenadas de forma decrescente e numericamente. Em caso contrário, a variável ***order*** vai apenas guardar ***"-n"*** que ordena de forma crescente e numericamente.

As verificações seguintes averigam qual opção de ordenação foi passada no ***argOpt***, de modo a imprimir no terminal os dados guardados em ***userInfo***, para cada utilizador, de acordo com o que foi introduzido como argumentos. Isto é feito com, além do ***printf***, o comando ***sort***, sendo que a opção ***"-k"*** designa o local onde operar, daí ser seguido por dois números (por exemplo ***-k2,2***), assegurando que a ordenação ocorre com a precedência da esquerda para a direita.

4 Comparação das estatísticas dos utilizadores

O script criado (*comparestats.sh*) compara dois ficheiros que salvaguardam a saída do programa *userstats.sh*. Após a sua execução, é possível observar a diferença entre os tempos de utilização e a diferença entre o número de sessões, considerando o primeiro ficheiro introduzido como o que representa os valores mais recentes. Os utilizadores que se encontram apenas num dos ficheiros também são apresentados.

4.1 Estrutura

Em semelhança ao script anterior, foram criados dois *Arrays Associativos* e dois *Arrays* para simplificar o tratamento dos dados.

```
3 # Arrays para guardar users e a sua informação
4 users1=() #Array para os user do 1º ficheiro
5 users2=() #Array para os user do 2º ficheiro
6 declare -A argOpt=() #Array Associativo onde são guardadas os argumento correspondentes às opções passadas
7 declare -A userInfo=() #Array Associativo onde é guardada a informação após o tratamento de dados correspondente a cada utilizador
```

Figure 9: *Arrays* mais relevantes na implementação deste script

- **users1** - *Array* onde são guardados os utilizadores, durante o tratamento de dados, correspondentes ao 1º ficheiro introduzido;
- **users2** - *Array* onde são guardados os utilizadores, durante o tratamento de dados, correspondentes ao 2º ficheiro introduzido;
- **argOpt** - *Array Associativo* onde são guardadas as opções e os respetivos argumentos durante o tratamentos de opções;
- **userInfo** - *Array Associativo* onde são guardadas as informações tratadas durante o tratamento de dados.

4.2 Tratamento de Opções

À semelhança do último script, as opções foram tratadas usando o comando shell *getopts*, abordado nas aulas práticas.

O tratamento de opções é iniciado com a chamada da função *args()*.

```
22 function args() {
23     repeat=0
24     while getopts rntai option; do
25         case "${option}" in
26             r);;
27             n | t | a | i)
28                 if [[ $repeat = 1 ]];then
29                     usage
30                 else
31                     repeat=1
32                 fi
33             ;;
34             *)
35                 usage
36             ;;
37             esac
38         argOpt[$option]="none"
39     done
40
41     if [ $((OPTIND+1)) -eq $# ]; then
42         eval input1=\${OPTIND}
43         eval input2=\${OPTIND + 1}
44     else
45         usage
46     fi
47
48     shift $((OPTIND - 1))
49 }
50
51
52
53
54
```

Figure 10: Função *args()* que procede ao tratamento de opções

O *getopts* recebe parâmetros (*rntai*) que correspondem às opções, neste caso sem argumentos, que o comando aceita. Estando este comando dentro de um *while*, é repetido as vezes necessárias para percorrer todas as opções e argumentos introduzidos no programa. As opções são guardadas, em cada ciclo, na variável *option* sendo executado o código associado a cada opção, consoante o *case statement*.

Para as opções "-n", "-t", "-a" e "-i" que não podem ser repetidas, foi criada uma variável *repeat* para averiguar se alguma destas variáveis já foi passada à função. Em caso afirmativo, é executada a função *usage()*.

Sempre que é introduzida uma opção inválida ou de forma incorreta, o *case statement* executa a função *usage()*, que indica a forma de utilização do script.

```

11 function usage() {
12     echo "Usage: $0 -r -n -t -a -i [ficheiro1] [ficheiro2]"
13     echo ""
14     echo "[ficheiro1] = Ficheiro mais recente para ser comparado"
15     echo "[ficheiro2] = Ficheiro mais antigo para ser comparado"
16     echo ""
17     exit
18 }

```

Figure 11: Função *usage()* que refere as opções e argumentos esperados

Ainda dentro do *while*, é adicionada a opção ao *Array Associativo argOpt*, se a função *usage()* não tiver sido acionada antes.

Após o ciclo, é executada a verificação se $\$((OPTIND+1))$ é igual ao número de argumentos passado à função. Visto que a variável *OPTIND* corresponde às opções e argumentos aceites no *getopts*, incluindo o nome do ficheiro, e *\$#* corresponde ao número de argumentos passados ao script, com exceção do nome do ficheiro, é **incrementado um** ao *OPTIND* de modo a comparar se o número de opções aceites no *getopts*, com o nome dos dois ficheiros de texto, é igual ao número de argumentos, executando a função *usage* em caso contrário. Assim, $\$((OPTIND))$ e $\$((OPTIND+1))$ correspondem, respetivamente, ao índice do argumento do primeiro e do segundo ficheiro.

Por fim, $\$((OPTIND-1))$ vai remover todas as opções que foram passadas pelo *getopts*, nos seus parâmetros. Desta forma, *\$1* vai referir o primeiro argumento passado ao script, que não é uma opção.

4.3 Leitura e tratamento de dados

Da mesma forma que a última implementação, o tratamento de dados é iniciado com a chamada da função *getUsers()*.

```
58 function getUsers() {
59     users1=$(cat $input1 | awk '{print $1}' | sort)
60     users2=$(cat $input2 | awk '{print $1}' | sort)
61     users=$(cat $users1[@] $users2[@])
62     unique_users=$(echo "$users" | tr ' ' '\n' | sort | uniq -u | tr '\n' ' ')
63 }
```

Figure 12: Função *getUsers()* que procede ao armazenamento e seleção dos utilizadores

A função começa por fazer a impressão dos conteúdos de ambos os ficheiros através do comando *cat*, sendo esta tratada pelo comando *awk* que armazena nos *Arrays* *users1* e *users2*, consoante o ficheiro, a primeira coluna da impressão. Visto que esta coluna corresponde aos utilizadores, estes *Arrays* vão possuir os utilizadores contidos em cada um dos ficheiros, ordenados crescentemente, como dita o comando *sort*.

Com o objetivo de criar um *Array* dos utilizadores que não estão repetidos em nenhum dos ficheiro, entenda-se, únicos, foi criado o *Array* temporário *users* onde estão combinados os utilizadores de ambos os ficheiros. Posteriormente, para chegar ao resultado pretendido, foi criado um *Array* *unique_users* que recebe a informação vinda da impressão do *Array* *users*, ordenada e apenas com os utilizadores únicos como é de esperar do comando *uniq*, juntamente com a opção *-u*, que não imprime as linhas repetidas. Tudo isto é auxiliado com a transformação de espaços por mudanças de linha *tr ' ' '\n'* para o comando *uniq -u* tratar os dados, repondo a formatação no final.

Em seguida, a informação obtida passa para *getUserInfo()*.

```
65 function getUserInfo() {
66     for user1 in ${users1[@]}; do
67         sessions1=$(cat $input1 | grep $user1 | awk '{print $2}')
68         total1=$(cat $input1 | grep $user1 | awk '{print $3}')
69         max1=$(cat $input1 | grep $user1 | awk '{print $4}')
70         min1=$(cat $input1 | grep $user1 | awk '{print $5}')
71     for user2 in ${users2[@]}; do
72         sessions2=$(cat $input2 | grep $user2 | awk '{print $2}')
73         total2=$(cat $input2 | grep $user2 | awk '{print $3}')
74         max2=$(cat $input2 | grep $user2 | awk '{print $4}')
75         min2=$(cat $input2 | grep $user2 | awk '{print $5}')
76         if [ "$user2" = "$user1" ]; then
77             sessions=$((sessions1 + sessions2))
78             total=$((total1 + total2))
79             max=$((max1 + max2))
80             min=$((min1 + min2))
81             userInfo[user2]=$(printf "%-8s %-5s %-6s %-5s %-5s\n" "$user2" "$sessions" "$total" "$max" "$min")
82         else
83             for unique in ${unique_users[@]}; do
84                 if [ "$unique" = "$user1" ]; then
85                     userInfo[user1]=$(printf "%-8s %-5s %-6s %-5s %-5s\n" "$user1" "$sessions1" "$total1" "$max1" "$min1")
86                 elif [ "$unique" = "$user2" ]; then
87                     userInfo[user2]=$(printf "%-8s %-5s %-6s %-5s %-5s\n" "$user2" "$sessions2" "$total2" "$max2" "$min2")
88                 fi
89             done
90         fi
91     done
92 done
93
94 printIt
95
96 }
```

Figure 13: Função *getUserInfo()* que trata e calcula os valores pretendidos na impressão

O tratamento de dados do script consiste em percorrer todos os elementos de ambos os *Arrays* criados na função

anterior, comparar os valores e, tratar os mesmos.

Isto é feito com dois ciclos *for* que percorrem os utilizadores guardados anteriormente nos *Arrays users1* e *users2*, guardando em *Arrays* o número de sessões, o tempo total das sessões, o tempo máximo das sessões e o tempo mínimo das sessões correspondente aos utilizadores atuais do ciclo. Alcançou-se este propósito imprimindo novamente os ficheiros através do comando *cat*, fazendo-se a seleção do utilizador atual do ciclo *for* com o comando *grep* e, por fim, a escolha da coluna correspondente aos dados em questão com o comando *awk*. A cada ciclo dos utilizadores do segundo ficheiro, é verificado se este utilizador é igual ao atual do primeiro ficheiro e, se isto se confirmar, são realizadas as subtrações dos valores do primeiro ficheiro com as do segundo, guardando essas informações no *Array Associativo userInfo*, à semelhança do último script. Em caso contrário, são percorridos os utilizadores únicos de modo a averiguar se este é pertencente ao primeiro ou ao segundo ficheiro, introduzindo os dados em *userInfo*, consoante a verificação.

4.4 Impressão ordenada dos dados filtrados

A impressão ordenada dos dados filtrados é feita com a chamada da função *printIt()* de igual forma ao script anterior.

```
98 function printIt() {
99     if [[ -v argOpt[r] ]]; then
100         # ordena decrescente (nome user)
101         order="-rn"
102     else
103         order="-n"
104     fi
105
106     if [[ -v argOpt[n] ]]; then
107         # ordenar por numero de sessoes
108         printf "%s\n" "${userInfo[@]}" | sort -k2,2 ${order}
109
110     elif [[ -v argOpt[t] ]]; then
111         # por tempo total
112         printf "%s\n" "${userInfo[@]}" | sort -k3,3 ${order}
113
114     elif [[ -v argOpt[a] ]]; then
115         # por tempo máximo
116         printf "%s\n" "${userInfo[@]}" | sort -k4,4 ${order}
117
118     elif [[ -v argOpt[i] ]]; then
119         # por tempo mínimo
120         printf "%s\n" "${userInfo[@]}" | sort -k5,5 ${order}
121
122     else
123         #ordem crescente (nome user)
124         printf "%s\n" "${userInfo[@]}" | sort -k1,1 ${order}
125     fi
126 }
```

Figure 14: Função *printIt()* que procede à impressão dos dados tratados consoante a ordem desejada

Em ambas as implementações, estas funções são chamadas no fim do script, permitindo obter os resultados desejados.

```
197  args "$@"  
198  getUsers  
199  getUserInfo
```

Figure 15: Chamada das várias funções em ambos os scripts

Com *args "\$@"* é chamada a função *args()*, juntamente com todos os argumentos passados no terminal, sendo que as outras duas funções são chamadas de seguida.

5 Resultados

Utilizando o computador da sala de aula, através de ligação remota, foram efetuados testes aos scripts desenvolvidos.

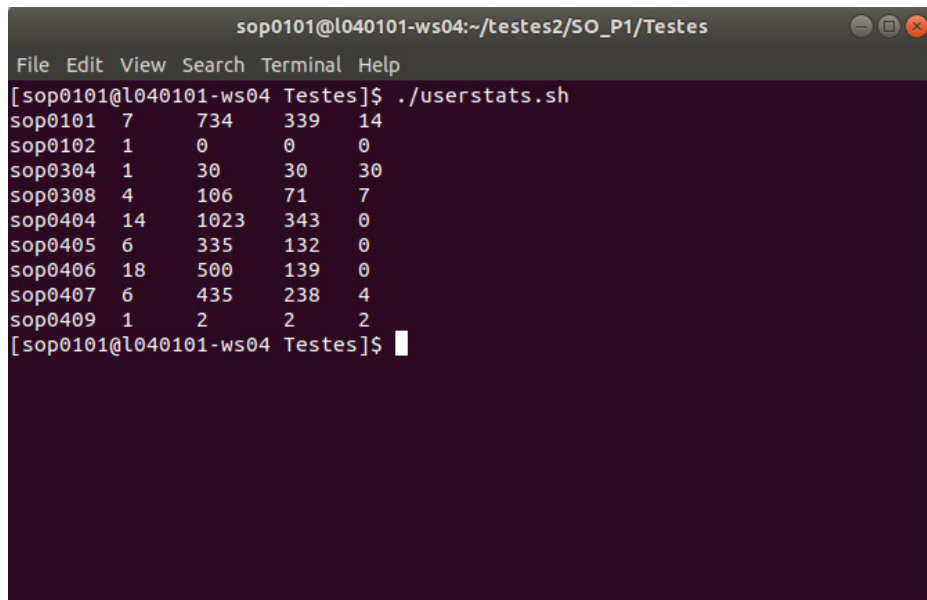
Ambos os scripts foram desenvolvidos a partir do comando *last* ou de ficheiros derivados com os dados tratados.

```
sop0101@l040101-ws04:~/Testes$ last
sop0101 pts/2      192.168.40.101    Sat Nov 30 01:33   still logged in
sop0405 pts/1      192.168.0.29     Sat Nov 30 01:08   still logged in
sop0405 pts/2      192.168.0.29     Sat Nov 30 00:23   - 00:23 (00:00)
sop0304 pts/1      192.168.0.45     Sat Nov 30 00:09   00:40 (00:30)
sop0406 pts/4      192.168.0.28     Fri Nov 29 22:44   22:50 (00:06)
sop0406 pts/3      192.168.0.28     Fri Nov 29 22:38   22:50 (00:11)
sop0406 pts/3      192.168.0.28     Fri Nov 29 22:20   22:38 (00:17)
sop0308 pts/2      192.168.0.42     Fri Nov 29 22:19   23:30 (01:11)
sop0406 pts/2      192.168.0.86     Fri Nov 29 22:04   22:10 (00:06)
sop0406 pts/2      192.168.0.86     Fri Nov 29 21:44   21:56 (00:06)
sop0406 pts/1      192.168.0.58     Fri Nov 29 20:52   23:03 (02:11)
sop0406 pts/1      192.168.0.58     Fri Nov 29 20:50   20:52 (00:01)
sop0406 pts/3      192.168.0.72     Fri Nov 29 17:56   20:16 (02:19)
sop0101 pts/2      192.168.52.57     Fri Nov 29 17:31   19:47 (02:15)
sop0101 pts/2      192.168.52.57     Fri Nov 29 17:07   17:27 (00:19)
sop0101 pts/1      192.168.52.57     Fri Nov 29 16:18   19:07 (02:49)
sop0101 pts/1      192.168.52.57     Fri Nov 29 16:04   16:18 (00:14)
sop0308 pts/1      192.168.55.128    Fri Nov 29 15:36   15:54 (00:18)
sop0308 pts/1      192.168.55.128    Fri Nov 29 15:21   15:32 (00:10)
sop0308 pts/1      192.168.55.128    Fri Nov 29 15:14   15:21 (00:07)
sop0101 pts/1      192.168.60.4      Fri Nov 29 03:16   04:15 (00:58)
sop0406 pts/2      192.168.55.120    Thu Nov 28 19:07   19:09 (00:02)
sop0406 pts/1      192.168.52.18     Thu Nov 28 18:55   19:07 (00:12)
sop0406 pts/1      192.168.52.18     Thu Nov 28 18:52   18:53 (00:00)
sop0406 pts/2      192.168.55.120    Thu Nov 28 18:36   19:00 (00:23)
sop0406 pts/2      192.168.52.18     Thu Nov 28 18:27   18:28 (00:01)
sop0406 pts/1      192.168.32.209    Thu Nov 28 17:36   18:45 (01:09)
sop0102 pts/1      192.168.0.73      Thu Nov 28 16:39   16:39 (00:00)
sop0406 pts/1      192.168.52.18     Thu Nov 28 15:33   15:35 (00:02)
sop0406 pts/1      192.168.52.18     Thu Nov 28 12:20   12:20 (00:00)
sop0405 pts/1      192.168.36.220    Wed Nov 27 20:29   22:41 (02:12)
sop0405 pts/2      192.168.54.45     Wed Nov 27 03:11   04:20 (01:08)
sop0405 pts/2      192.168.54.45     Wed Nov 27 01:00   03:10 (02:10)
sop0101 pts/1      192.168.51.43     Tue Nov 26 23:24   05:03 (05:39)
sop0407 pts/1      192.168.0.33      Mon Nov 25 13:54   14:04 (00:09)
sop0404 pts/1      192.168.33.125    Mon Nov 25 10:23   10:23 (00:00)
sop0404 pts/1      192.168.33.125    Mon Nov 25 10:06   10:23 (00:14)
sop0404 pts/1      192.168.0.86      Sun Nov 24 22:09   22:11 (00:01)
sop0404 pts/1      192.168.0.86      Sun Nov 24 21:56   21:57 (00:01)
sop0404 pts/1      192.168.0.86      Sun Nov 24 17:54   21:49 (03:54)
sop0407 pts/1      192.168.33.99     Sat Nov 23 19:45   19:50 (00:04)
sop0407 pts/2      192.168.33.99     Sat Nov 23 16:08   17:22 (01:14)
sop0406 pts/1      192.168.57.142    Sat Nov 23 16:07   16:29 (00:21)
sop0404 pts/1      192.168.0.51      Sat Nov 23 14:26   14:33 (00:06)
sop0404 pts/1      192.168.0.51      Sat Nov 23 12:26   13:52 (01:25)
sop0407 pts/1      192.168.0.68      Sat Nov 23 03:35   04:51 (01:16)
sop0407 pts/1      192.168.0.68      Fri Nov 22 23:37   03:35 (03:58)
sop0407 pts/1      192.168.0.68      Fri Nov 22 23:02   23:36 (00:34)
sop0404 pts/2      192.168.0.70      Fri Nov 22 21:08   21:15 (00:06)
sop0404 pts/1      192.168.0.50      Fri Nov 22 20:36   22:47 (02:11)
sop0405 pts/1      192.168.44.104    Fri Nov 22 19:15   19:20 (00:05)
sop0404 pts/1      192.168.57.10     Fri Nov 22 12:31   15:39 (03:08)
sop0202 pts/0      :0                Fri Nov 22 10:26   still logged in
reboot system boot 4.4.5-300.fc23.x Fri Nov 22 10:23   still running
reboot system boot 4.4.5-300.fc23.x Fri Nov 22 10:23   still running
sop0409 pts/2      192.168.52.255    Thu Nov 21 17:18   17:21 (00:02)
sop0404 pts/2      192.168.54.101    Thu Nov 21 10:51   16:34 (05:43)
sop0404 pts/2      192.168.54.101    Thu Nov 21 10:47   10:50 (00:03)
sop0404 pts/3      192.168.54.101    Thu Nov 21 10:41   10:46 (00:04)
sop0404 pts/2      172.17.13.6       Thu Nov 21 10:34   10:41 (00:07)
sop0406 pts/2      192.168.46.139    Wed Nov 20 16:27   17:20 (00:53)

wtmp begins Wed Nov 20 16:27:19 2019
[sop0101@l040101-ws04:~/Testes]$
```

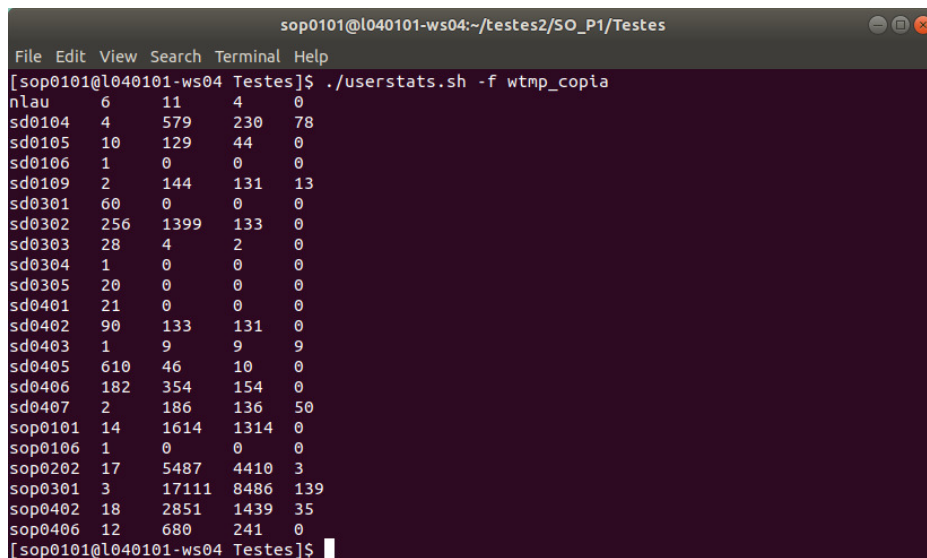
Figure 16: Execução do comando *last* no computador da sala de aula, por volta das horas em que foram realizados os testes

5.1 Estatísticas dos utilizadores



```
sop0101@l040101-ws04:~/testes2/SO_P1/Testes
File Edit View Search Terminal Help
[sop0101@l040101-ws04 Testes]$ ./userstats.sh
sop0101 7 734 339 14
sop0102 1 0 0 0
sop0304 1 30 30 30
sop0308 4 106 71 7
sop0404 14 1023 343 0
sop0405 6 335 132 0
sop0406 18 500 139 0
sop0407 6 435 238 4
sop0409 1 2 2 2
[sop0101@l040101-ws04 Testes]$
```

Figure 17: Execução do script sem nenhum argumento



```
sop0101@l040101-ws04:~/testes2/SO_P1/Testes
File Edit View Search Terminal Help
[sop0101@l040101-ws04 Testes]$ ./userstats.sh -f wtmp_copia
nlau 6 11 4 0
sd0104 4 579 230 78
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 5487 4410 3
sop0301 3 17111 8486 139
sop0402 18 2851 1439 35
sop0406 12 680 241 0
[sop0101@l040101-ws04 Testes]$
```

Figure 18: Execução do script com a opção `-f` e a cópia do ficheiro `/var/log/wtmp`, do computador da sala, (*wtmp_copia*) relativo a outra data

```
sop0101@l040101-ws04:~/testes2/SO_P1/Testes
File Edit View Search Terminal Help
[sop0101@l040101-ws04 Testes]$ ./userstats.sh -u "sop.*"
sop0101 7 734 339 14
sop0102 1 0 0 0
sop0304 1 30 30 30
sop0308 4 106 71 7
sop0404 14 1023 343 0
sop0405 6 335 132 0
sop0406 18 500 139 0
sop0407 6 435 238 4
sop0409 1 2 2 2
[sop0101@l040101-ws04 Testes]$
```

Figure 19: Execução do script com a opção `-u` e o nome dos utilizadores para serem verificados através de uma expressão regular, neste caso `"sop.*"`

```
sop0101@l040101-ws04:~/testes2/SO_P1/Testes
File Edit View Search Terminal Help
[sop0101@l040101-ws04 Testes]$ ./userstats.sh -g sop
sop0101 7 734 339 14
sop0102 1 0 0 0
sop0304 1 30 30 30
sop0308 4 106 71 7
sop0404 14 1023 343 0
sop0405 6 335 132 0
sop0406 18 500 139 0
sop0407 6 435 238 4
sop0409 1 2 2 2
[sop0101@l040101-ws04 Testes]$
```

Figure 20: Execução do script com a opção `-g` e o grupo a ser seleccionado, `sop`

```
sop0101@l040101-ws04:~/testes2/SO_P1/Testes
File Edit View Search Terminal Help
[sop0101@l040101-ws04 Testes]$ ./userstats.sh -s "Nov 28 10:00" -e "Nov 29 10:00"
sop0101 1 58 58 58
sop0102 1 0 0 0
sop0406 8 109 69 0
[sop0101@l040101-ws04 Testes]$
```

Figure 21: Execução do script com a opção `-s` e `-e` no intervalo de "Nov 28 10:00" a "Nov 29 10:00"

```
sop0101@l040101-ws04:~/testes2/SO_P1/Testes
File Edit View Search Terminal Help
[sop0101@l040101-ws04 Testes]$ ./userstats.sh -t -u "sop.*"
sop0102 1 0 0 0
sop0409 1 2 2 2
sop0304 1 30 30 30
sop0308 4 106 71 7
sop0405 6 335 132 0
sop0407 6 435 238 4
sop0406 18 500 139 0
sop0101 7 734 339 14
sop0404 14 1023 343 0
[sop0101@l040101-ws04 Testes]$
```

Figure 22: Execução do script com a opção `-t` e `-u` com o nome dos utilizadores para serem verificados através de uma expressão regular, neste caso `"sop.*"`

```
sop0101@l040101-ws04:~/testes2/SO_P1/Testes
File Edit View Search Terminal Help
[sop0101@l040101-ws04 Testes]$ ./userstats.sh -n -u "sop.*"
sop0102 1 0 0 0
sop0304 1 30 30 30
sop0409 1 2 2 2
sop0308 4 106 71 7
sop0405 6 335 132 0
sop0407 6 435 238 4
sop0101 7 734 339 14
sop0404 14 1023 343 0
sop0406 18 500 139 0
[sop0101@l040101-ws04 Testes]$
```

Figure 23: Execução do script com a opção `-n` e `-u` com o nome dos utilizadores para serem verificados através de uma expressão regular, neste caso `"sop.*"`

```
sop0101@l040101-ws04:~/testes2/SO_P1/Testes
File Edit View Search Terminal Help
[sop0101@l040101-ws04 Testes]$ ./userstats.sh -t -r -u "sop.*"
sop0404 14 1023 343 0
sop0101 7 734 339 14
sop0406 18 500 139 0
sop0407 6 435 238 4
sop0405 6 335 132 0
sop0308 4 106 71 7
sop0304 1 30 30 30
sop0409 1 2 2 2
sop0102 1 0 0 0
[sop0101@l040101-ws04 Testes]$
```

Figure 24: Execução do script com a opção `-t`, `-r` e `-u` com o nome dos utilizadores para serem verificados através de uma expressão regular, neste caso `"sop.*"`

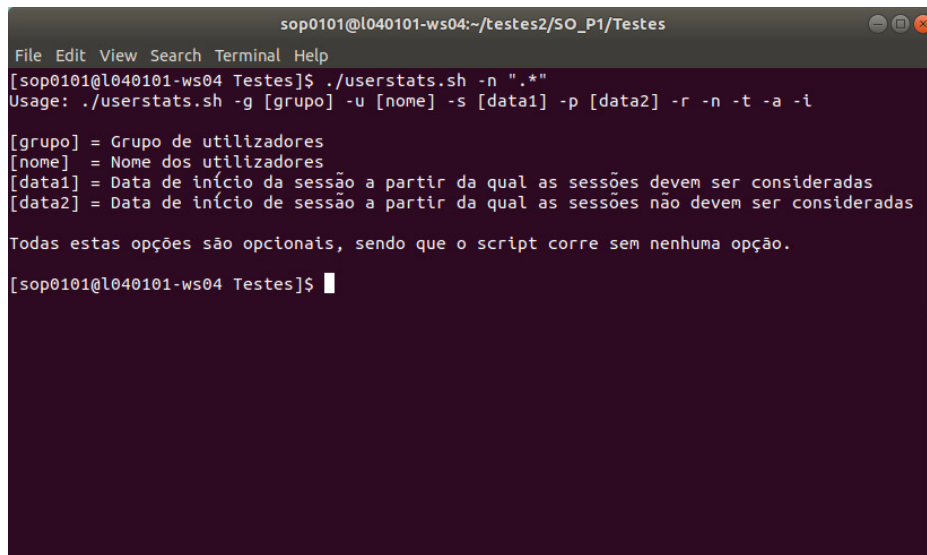

```
sop0101@l040101-ws04:~/testes2/SO_P1/Testes
File Edit View Search Terminal Help
[sop0101@l040101-ws04 Testes]$ ./userstats.sh -a -r -u "sop.*"
sop0404 14 1023 343 0
sop0101 7 734 339 14
sop0407 6 435 238 4
sop0406 18 500 139 0
sop0405 6 335 132 0
sop0308 4 106 71 7
sop0304 1 30 30 30
sop0409 1 2 2 2
sop0102 1 0 0 0
[sop0101@l040101-ws04 Testes]$
```

Figure 25: Execução do script com a opção `-a`, `-r` e `-u` com o nome dos utilizadores para serem verificados através de uma expressão regular, neste caso `"sop.*"`

```
sop0101@l040101-ws04:~/testes2/SO_P1/Testes
File Edit View Search Terminal Help
[sop0101@l040101-ws04 Testes]$ ./userstats.sh -i -r -u "sop.*"
sop0304 1 30 30 30
sop0101 7 734 339 14
sop0308 4 106 71 7
sop0407 6 435 238 4
sop0409 1 2 2 2
sop0406 18 500 139 0
sop0405 6 335 132 0
sop0404 14 1023 343 0
sop0102 1 0 0 0
[sop0101@l040101-ws04 Testes]$
```

Figure 26: Execução do script com a opção `-i`, `-r` e `-u` com o nome dos utilizadores para serem verificados através de uma expressão regular, neste caso `"sop.*"`

No próximo teste foi introduzido um argumento juntamente com a opção *-n*, que não recebe argumentos e, como é possível verificar, o tratamento de opções do script facilmente detetou o erro e chamou a função *usage*.



```
sop0101@l040101-ws04:~/testes2/SO_P1/Testes
File Edit View Search Terminal Help
[sop0101@l040101-ws04 Testes]$ ./userstats.sh -n ".*"
Usage: ./userstats.sh -g [grupo] -u [nome] -s [data1] -p [data2] -r -n -t -a -i

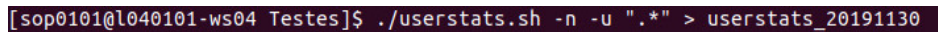
[grupo] = Grupo de utilizadores
[nome]   = Nome dos utilizadores
[data1]  = Data de início da sessão a partir da qual as sessões devem ser consideradas
[data2]  = Data de início de sessão a partir da qual as sessões não devem ser consideradas

Todas estas opções são opcionais, sendo que o script corre sem nenhuma opção.

[sop0101@l040101-ws04 Testes]$
```

Figure 27: Execução do script com a opção *-n* e o argumento *".*"*

Por fim, foi guardado o *output* deste script, de modo a ser possível utilizá-lo no script seguinte.



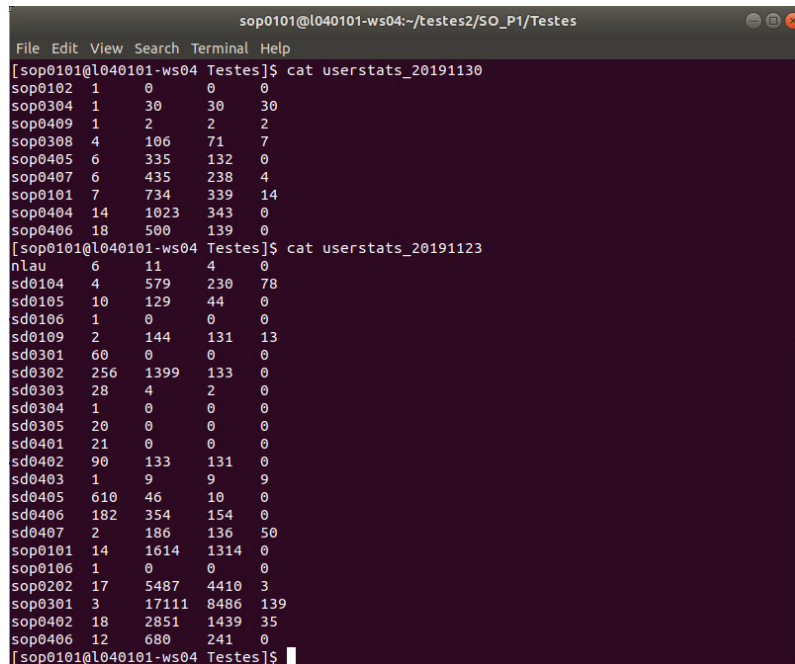
```
[sop0101@l040101-ws04 Testes]$ ./userstats.sh -n -u ".*" > userstats_20191130
```

Figure 28: Execução do script com a opção *-n* e *-u* com o nome dos utilizadores para serem verificados através de uma expressão regular, neste caso *"sop.*"*, redirecionando o output para o ficheiro *userstats_20191130*

5.2 Comparação das estatísticas dos utilizadores

Os testes da comparação das estatísticas dos utilizadores foram realizados recorrendo aos outputs do script anterior. Foram utilizados ficheiro datados de 23/11/2019 e 30/11/2019, sendo que foi considerado o primeiro aquele que possuía os valores mais recentes e portanto, as diferenças foram calculadas entre o primeiro e o segundo ficheiro.

A figura seguinte mostra o conteúdo de ambos os ficheiros, permitindo a sua análise antes da execução do script.



```
sop0101@l040101-ws04:~/testes2/SO_P1/Testes
File Edit View Search Terminal Help
[sop0101@l040101-ws04 Testes]$ cat userstats_20191130
sop0102 1 0 0 0
sop0304 1 30 30 30
sop0409 1 2 2 2
sop0308 4 106 71 7
sop0405 6 335 132 0
sop0407 6 435 238 4
sop0101 7 734 339 14
sop0404 14 1023 343 0
sop0406 18 500 139 0
[sop0101@l040101-ws04 Testes]$ cat userstats_20191123
nlau 6 11 4 0
sd0104 4 579 230 78
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 5487 4410 3
sop0301 3 17111 8486 139
sop0402 18 2851 1439 35
sop0406 12 680 241 0
[sop0101@l040101-ws04 Testes]$
```

Figure 29: Impressão do conteúdo dos ficheiro *userstats_20191123* e *userstats_20191130*

É possível observar que apenas existem dois utilizadores iguais, em ambos os ficheiros. Desta forma, é possível prever o conteúdo que este script imprimirá, sendo que apenas efetuará os cálculos nestes utilizadores, mantendo os dados dos outros.

```
sop0101@l040101-ws04:~/testes2/SO_P1/Testes
File Edit View Search Terminal Help
[sop0101@l040101-ws04 Testes]$ ./comparestats.sh userstats_20191123 userstats_20191130
nlau 6 11 4 0
sd0104 4 579 230 78
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 7 880 975 -14
sop0102 1 0 0 0
sop0106 1 0 0 0
sop0202 17 5487 4410 3
sop0301 3 17111 8486 139
sop0304 1 30 30 30
sop0308 4 106 71 7
sop0402 18 2851 1439 35
sop0404 14 1023 343 0
sop0405 6 335 132 0
sop0406 -6 180 102 0
sop0407 6 435 238 4
sop0409 1 2 2 2
[sop0101@l040101-ws04 Testes]$
```

Figure 30: Execução do script com os argumentos *userstats_20191123* e *userstats_20191130*

```
sop0101@l040101-ws04:~/testes2/SO_P1/Testes
File Edit View Search Terminal Help
[sop0101@l040101-ws04 Testes]$ ./comparestats.sh -r userstats_20191123 userstats_20191130
sop0409 1 2 2 2
sop0407 6 435 238 4
sop0406 -6 180 102 0
sop0405 6 335 132 0
sop0404 14 1023 343 0
sop0402 18 2851 1439 35
sop0308 4 106 71 7
sop0304 1 30 30 30
sop0301 3 17111 8486 139
sop0202 17 5487 4410 3
sop0106 1 0 0 0
sop0102 1 0 0 0
sop0101 7 880 975 -14
sd0407 2 186 136 50
sd0406 182 354 154 0
sd0405 610 46 10 0
sd0403 1 9 9 9
sd0402 90 133 131 0
sd0401 21 0 0 0
sd0305 20 0 0 0
sd0304 1 0 0 0
sd0303 28 4 2 0
sd0302 256 1399 133 0
sd0301 60 0 0 0
sd0109 2 144 131 13
sd0106 1 0 0 0
sd0105 10 129 44 0
sd0104 4 579 230 78
nlau 6 11 4 0
[sop0101@l040101-ws04 Testes]$
```

Figure 31: Execução do script com a opção *-r* os argumentos *userstats_20191123* e *userstats_20191130*

```
sop0101@l040101-ws04:~/testes2/SO_P1/Testes
File Edit View Search Terminal Help
[sop0101@l040101-ws04 Testes]$ ./comparestats.sh -t userstats_20191123 userstats_20191130
sd0106 1 0 0 0
sd0301 60 0 0 0
sd0304 1 0 0 0
sd0305 20 0 0 0
sd0401 21 0 0 0
sop0102 1 0 0 0
sop0106 1 0 0 0
sop0409 1 2 2 2
sd0303 28 4 2 0
sd0403 1 9 9 9
nlau 6 11 4 0
sop0304 1 30 30 30
sd0405 610 46 10 0
sop0308 4 106 71 7
sd0105 10 129 44 0
sd0402 90 133 131 0
sd0109 2 144 131 13
sop0406 -6 180 102 0
sd0407 2 186 136 50
sop0405 6 335 132 0
sd0406 182 354 154 0
sop0407 6 435 238 4
sd0104 4 579 230 78
sop0101 7 880 975 -14
sop0404 14 1023 343 0
sd0302 256 1399 133 0
sop0402 18 2851 1439 35
sop0202 17 5487 4410 3
sop0301 3 17111 8486 139
[sop0101@l040101-ws04 Testes]$
```

Figure 32: Execução do script com a opção *-t* os argumentos *userstats_20191123* e *userstats_20191130*

6 Conclusão

Em suma, os objetivos propostos pelo docente foram alcançados de acordo com as indicações dadas. A implementação foi pensada de forma a obter todos os resultados pretendidos, sendo que não houve grande incidência na otimização de código, uma vez que não era um dos requisitos. Ainda assim, tivemos em consideração, e tentámos, encontrar sempre soluções que não exigissem muito tempo de espera pelo output final. Considera-se, portanto, que a realização deste trabalho prático foi um sucesso.

Surgiram várias advertências pelo caminho, nomeadamente devido à especificidade da linguagem Bash, sendo que algumas das vezes se tornou complicado resolver, ou até mesmo encontrar, o problema em questão. Por vezes o problema era apenas falta de aspas em torno das variáveis, espaços nos sítios incorretos ou apenas um ciclo *for* que não chegou a ser terminado, mas houve definitivamente vezes onde a procura de soluções chegou a demorar algumas horas. No entanto, tudo se encontra a funcionar mediante o trabalho proposto, sendo que o grupo acha que estas pequenas barreiras que foram aparecendo os obrigou a pesquisar e, consequentemente, a adquirir novos e mais vastos conhecimentos sobre diversos assuntos relacionados com esta linguagem.

Para além do referido, existiu outra advertência que condicionou a fase de testes. O grupo concluiu, ainda, que as configurações de ssh para as máquinas da sala de aula não se encontravam bem implementadas, impossibilitando a ligação aos computadores através de uma VPN da rede da Universidade. Deste modo, a realização dos testes teve de ser sempre realizada dentro da Universidade, trazendo alguns transtornos quando o trabalho se encontrava a ser desenvolvido fora da mesma.

Quanto às estruturas de dados, concluiu-se que o uso de *Arrays Associativos* em vez de *Arrays* normais acabou por ser mais vantajoso, facilitando a manipulação dos dados.

Por fim, foi também concluído que existem vários utilizadores a recorrer ao computador em questão, maioritariamente ao final do dia e com sessões relativamente pequenas (menos de uma hora). Os grupos de utilizadores correspondentes a alunos, acabam por operar com muita frequência, devido à necessidade que estes têm para resolver questões escolares.

7 Bibliografia

- [1] <https://pplware.sapo.pt/linux/personalize-a-prompt-de-comandos-da-Bash-no-linux/>
- [2] <https://www.shellscript.sh/tips/getopts/>
- [3] <https://www.gnu.org/software/Bash/manual/Bash.html>
- [4] <https://aurelio.net/shell/canivete/>
- [5] <https://stackoverflow.com>