universidade de aveiro

Sistemas Operativos
Prof. Nuno Lau

ESTATÍSTICAS DE UTILIZADORES EM BASH

Alexandre Rodrigues - 92951
Rui Fernandes - 92952

A participação é repartida de igual forma.

Índice

Introdução

No âmbito da disciplina Sistemas Operativos, o presente trabalho tem como principal objetivo o "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. Permitem também comparar os dados obtidos em em períodos distintos."

(excerto presente no enunciado do projeto)

Ao longo do relatório irá ser apresentada a metedologia utilizada para solucionar o problema do userstats.sh e o problema do comparestats.sh. Para complementar, estão presentes as etapas e as ideias que usamos para contruir todas as funções e, todos os tratamentos de erros e todos os testes realizados para comprovar que tudo foi verificado e analisado.

Metodologia usada para Solucionar o Problema

Userstats.sh

```
#!/bin/bash
declare -A users=()     #All users
declare -A args=()      #Global arguments array
declare times=()        #Array for use in calcTime()
declare -A info=()      #Associative array with users as keys and the relevant info as values
declare init_date=19700101010000    #Initial date
declare final_date=50000101010000   #Final date
declare num_order_opts=0            #number of ordering options (-n -t -a -i)
```

Fig. 1 – Declaração de arrays e variáveis.

Todos os arrays utilizados para armazenar informação relevante, declarões de início de data (1 de janeiro de 1970, Era Unix ou Posix Time) e fim de data com valor elevado, e ainda o número de ocorrencias de opções de ordenação.

Validação dos argumentos

```
argForm() {
    if [[ $1 =~ -.* ]]; then
        echo "ERROR: argument format not valid"
        exit
    fi
}
```

Fig. 2 – Função *argForm()*

A função **argForm()** foi criada para evitar repetição de código na função parse_arguments(), verifica se o input da função verifica a regex "-.*", ou seja se começa pelo caracter "-". Isto é usado no contexto da parse_arguments para o caso de uma opção que requer um argumento ser passada sem um argumento e seguida de outra opção, o getopts assume que a segunda opção é o argumento da primeira, a função **argForm()** lança uma mensagem de erro caso isto se verifique.

```
parse_arguments () {
    while getopts ":rntaig:u:f:s:e:" opt; do
        case $opt in
            r)
                argForm "$OPTARG"
            ;;
            n|t|a|i)
                argForm "$OPTARG"
                if [[ num_order_opts -gt 0 ]]; then
                    echo "ERROR: too many ordering options, use only one of the following: -n -t -a -i"
                    exit
                fi
                num_order_opts=$((num_order_opts+1))
            ;;
            u|g)
                argForm "$OPTARG"
            ;;
            f)
                argForm "$OPTARG"

                if [[ ! -f "$OPTARG" ]]; then
                    echo "ERROR: file not found!"
                    exit
                fi
            ;;
            s|e)
                argForm "$OPTARG"

                # converts argument date to number of seconds since 1970-01-01 00:00:00 UTC
                OPTARG=$(date -d "$OPTARG" +%s)

            ;;
```

```
            ;;
            \?)
                echo "ERROR: invalid option: -$OPTARG"
                exit
                ;;
            :)
                echo "ERROR: option -$OPTARG needs an argument."
                exit
                ;;
            *)
                echo "ERROR: option not implemented: -$OPTARG"
                exit
            ;;
        esac
        if [[ $OPTARG == "" ]]; then
            args[$opt]="$opt"
        else
            args[$opt]=$OPTARG # save all options in array
        fi
    done

    shift $((OPTIND - 1))
    if [[ -n "$@" ]]; then
        echo "ERROR: parameters are not valid: $@"
        exit
    fi
}
```

Fig. 3 – Função *parse_arguments()*

Função ***parse_arguments()*** usada para a validação de todos os argumentos/opções inseridas pelo Utilizador. O Utilizador consegue inserir todas as opções **válidas**, presentes em "rntai:g:u:f:s:e:", de modo que, a sua leitura é feita de forma sequencial e a ordem como são colocadas não é estática, ou seja, a ordem como as mesmas são inseridas pode variar. À medida que as opções e argumentos são validados, são armazenados no array associativo **args[]** cujas **keys** são os caracteres das opções e os **values** os argumentos associados (no caso de opções sem argumentos é guardado um valor simbólico), este array vai servir para verificar que opções foram passadas no programa.

Caso seja passada mais que uma opção de ordenação da informação (-n -t -a -i), o programa é interrompido e é impressa uma mensagem de erro. Caso seja passada a opção -f é verificado se o ficheiro existe, e quando estão presentes as opções -e ou -s os seus argumentos são convertidos de datas para número de segundos.

Caso seja inserida uma opção não suportada, é apresentada a mensagem "Invalid option: -$OPTARG", caso uma opção com argumento obrigatório seja passada sem argumento, é apresentada a mensagem "Option -$OPTARG needs an argument.", se por alguma razão na validação dos argumentos surja uma opção não suportada pelo case será mostrada a seguinte mensagem de erro: "Option not implemented: -$OPTARG" e, finalmente, caso sejam passados argumentos não requeridos por opções, é apresentada a mensagem "Parameters are not valid: $@".

Função calcTime()

```
function calcTime(){
            t=$1

            if (( ${#t} == 5 )); then

                mins=$(echo $t | awk -F: '{ print ($1 * 60) + $2}')
                times[t_index]=$mins
                total_time=$(($total_time + $mins))

            else
                mins=$(echo $t | tr '+' ':' | awk -F: '{ print ($1 * 1440) + ($2 * 60) + $3}')
                times[t_index]=$mins
                total_time=$(($total_time + $mins))
            fi
}
```

Fig. 4 – Função *calcTime()*

Função utilizada para converter o tempo dos formatos HH:MM e D+HH:MM para tempo em minutos e calcular o respetivo tempo total para cada utilizador

Tratamento dos argumentos

Opções *-s* e *-e*

```
169 parse_arguments "$@"
170
171 if [[ ${args['s']} ]]; then
172     init_date=$(date --date=@${args['s']} "+%Y%m%d%H%M%S")
173 fi
174 if [[ ${args['e']} ]]; then
175     final_date=$(date --date=@${args['e']} "+%Y%m%d%H%M%S")
176 fi
177 getUsers
178 getInfo
179 printInfo
```

Fig. 5 – Opções -s e -e

Após ser chamada a função **parse_arguments()** é verificada a presença das opções -s e -e, no caso de estarem presentes as respetivas datas de filtragem são alteradas para os argumentos passados com as opções. De seguida são chamadas as funções **getUsers()**, **getInfo()** e **printInfo().**

Função getUsers() e opção -f

```
function getUsers() {

    if [[ ${args['f']} ]]; then
        users=$(last -f ${args['f']} -s "$init_date" -t "$final_date" | awk '{print $1}' | sort -u | head -n -1 | sort | sed '/reboot/d' | sed "/${args['f']
    else
        users=$(last -s "$init_date" -t "$final_date" | awk '{print $1}' | sort -u | head -n -1 | sort |sed '/reboot/d' | sed '/wtmp/d'|grep -v '^$'z1)
    fi

    users=$(echo $users| tr " " "\n")
}
```

Fig. 6 – Função *getUsers()*

Inicialmente, para obter todos os utilizadores relevantes, utilizamos a função ***getUsers()***. Os utilizadores são recolhidos através do comando last chamado com as datas de filtragem, caso esteja presente a opção -f o comando last é executado usando o ficheiro passado como argumento ao invés do default : /var/log/wtmp. É de mencionar que é apenas recolhida a informação da primeira coluna do comando last (awk '{print $1}') e que são removidas as linhas indesejadas através dos vários comandos sed. Os utilizadores recolhidos são armazenados no array **users[]** para serem utilizados na função **getInfo().**

Função getInfo() e opção -f

```
function getInfo() {
    for user in $users
    do
        if [[ ${args['f']} ]]; then
            time=$(last -f ${args['f']} -s "$init_date" -t "$final_date" | grep $user | awk '{print $10}' | sed '/in/d' | sed 's/[)(]//g' )
            session=$(last -f ${args['f']} -s "$init_date" -t "$final_date" | awk '{print $1}' | grep $user | wc -l)
        else
            time=$(last -s "$init_date" -t "$final_date" | grep $user | awk '{print $10}'  |sed '/in/d' | sed 's/[)(]//g')
            session=$(last -s "$init_date" -t "$final_date" | awk '{print $1}' | grep $user | wc -l)
        fi

        total_time=0
        t_index=0
        for t in $time; do
            calcTime "$t"
            t_index=$((t_index+1))
        done
        IFS=$'\n'
        max_time=$(echo "${times[*]}" | sort -nr | head -n1)
        min_time=$(echo "${times[*]}" | sort -nr | tail -1)
        info[$user]="$user $session $total_time $max_time $min_time"
        unset times #destroi array para o proximo user

    done
    unset times #array fica livre
}
```

Fig. 7 – Função *getInfo()*

A função **getInfo()** tem como objetivo recolher a informação pretendida de cada user, esta é: o número de sessões, e os tempos: total, máximo e mínimo de ligação.

Percorrem-se os utilizadores do array **users[]** e utiliza-se de novo o comando last para obter a informação relevante, as opções -f, -s e -e são lidadas da mesma maneira que na função **getUsers().**

Para se obter o numero de sessões, guardado na variável session, analiza-se a primeira coluna do last (awk '{print $1}'), recolhe-se apenas os dados referentes ao dado utilizador (grep $user) e contam-se o número de ocorrências (wc -l).

A variável time guarda todos os tempos de sessão ainda no formato HH:MM (ou D+HH:MM), recolhe a informação do dado utilizador (grep $user), analiza a décima coluna (awk '{print $10}'), e retira os parênteses (sed 's/[)(]//g'). O comando sed '/in/d' serve para não considerar a linha que se refere á

sessão corrente. De seguida percorrem-se os elementos de time e calcula-se o tempo total, máximo e mínimo com auxílio da função **calcTime().**

A informação é guardada no array associativo info que tem como **keys** os utilizadores e como **values** um string que contém a informação formatada.

Função printInfo() e opções -u -g -r -n -t -a -i

```
function printInfo() {

    if [[ ${args['u']} ]]; then
        for user in $users; do
            if [[ ! $user =~ ${args['u']} ]]; then
                unset info[$user]
            fi
        done
    fi

    if [[ ${args['g']} ]]; then
        for user in $users; do
            if  ! (id -nG "$user" | grep -qw "${args['g']}"); then
                unset info[$user]
            fi
        done
    fi

    order=""
    if [[ ${args['r']} ]]; then
        order="r"
    fi

    if [[ ${args['n']} ]]; then
        printf "%s\n" "${info[@]}" | sort -k2,2n$order
    elif [[ ${args['t']} ]]; then
        printf "%s\n" "${info[@]}" | sort -k3,3n$order
    elif [[ ${args['a']} ]]; then
        printf "%s\n" "${info[@]}" | sort -k4,4n$order
    elif [[ ${args['i']} ]]; then
        printf "%s\n" "${info[@]}" | sort -k5,5n$order
    else
        printf "%s\n" "${info[@]}" | sort -k1,1$order
    fi
}
```

Fig. 8 – Função *getUsers()*

A função ***printInfo()*** trata das opções -u, -g, -r, -n, -t, -a, - i e imprime a informação formatada.

Na presença da opção -u percorrem-se os utilizadores e retira-se do array os que não correspondem á expressão regex passada como argumento.

Na presença da opção -g percorrem-se os utilizadores e retiram-se os que não pertencem ao grupo passado como argumento. O comando id -nG “$user” lista os grupos a que o user pertence e o comando grep -qw “${args[‘g’]}” verifica se o grupo está no conjunto.

Consoante a opção de ordenação selecionada (-n, -t, -a ou -i) a informação é impressa ordenada pela coluna respetiva usando o comando sort -k,n.

Quando está presente a opção -r é acrescentado ao comando sort de modo à ordem ficar reversa.

Metodologia usada para Solucionar o Problema

CompareStats.sh

```bash
#!/bin/bash

declare -A args=()
declare -A linesA=()
declare -A linesB=()
declare -A final_info=()
declare fileA=""
declare fileB=""
declare num_order_opts=0    #number of ordering options (-n -t -a -i)
```

Fig. 9 – Fig. 1 – Declaração de arrays

Todos os arrays utilizados para armazenar informação relevante, declarões de ficheiros e o número de ocorrencias de opções de ordenação.

```bash
parse_arguments () {
    while getopts ":rntai" opt; do
        case $opt in
            r)
            ;;
            n|t|a|i)
                if [[ num_order_opts -gt 0 ]]; then
                    echo "ERROR: too many ordering options, use only one of the following: -n -t -a -i"
                    exit
                fi
                num_order_opts=$((num_order_opts+1))
            ;;
            \?)
                echo "Invalid option: -$OPTARG"
                exit
                ;;
            *)
                echo "Option not implemented: -$OPTARG"
                exit
            ;;
        esac
        args[$opt]="$opt"
    done

    shift $((OPTIND - 1))
    if [[ -n "$@" ]]; then
        if [[ $# -ge 3 ]] || [[ $# -le 1 ]]; then
            echo "ERROR: The program requires exactly 2 files!"
            exit
        else
            if [[ ! -f "$1" ]]; then
                echo "ERROR: file $1 not found!"
                exit
            fi
            if [[ ! -f "$2" ]]; then
                echo "ERROR: file $2 not found!"
                exit
            fi
            fileA="$1"
            fileB="$2"
        fi
    else
        echo "ERROR: The program requires exactly 2 files!"
        exit
    fi
}
```

Fig. 10 – Função parse_arguments()

Função ***parse_arguments()*** usada para validar as opções -r (ordem decrescente) -n (por número de sessões) -t (por tempo total) -a (por tempo máximo) -i (por tempo mínimo). Caso se verifique que foi passada mais de uma das opções de ordenação (não inclui a -r) aparece a mensagem de erro "*ERROR: too many ordering options, use only one of the following: -n -t -a -i*". Caso a opção seja inválida aparece uma mensagem de erro.
De seguida se foram passados menos ou mais de 2 ficheiros imprime uma mensagem de erro ("ERROR: The program requires exactly 2 files!"). Caso alguma das 2 files não exista é também lançada uma mensagem de erro ("ERROR: file not found!")

```
while read lineA
   do
   userA=$(echo $lineA  | awk '{print $1}')
   linesA[$userA]=$lineA

done < $fileA

while read lineB
   do
   userB=$(echo $lineB  | awk '{print $1}')
   linesB[$userB]=$lineB

done < $fileB
```

Fig. 11 – Leituras dos ficheiros

Atribuição de todos os utilizadores do ficheiro do **fileA** a um array **userA**. De seguida, para cada chave do **linesA** é atribuído o valor completo da **linhaA** que contem tudo o que o **userstats** devolve, ou seja, número de sessões, tempo total de ligação (em minutos), duração máxima e duração mínima das sessões dos utilizadores. O mesmo ocorre para o outro ficheiro intitulado como **fileB.**

```
for user in "${!linesA[@]}"
do
    if [[ ${linesB[$user]} ]]; then
        difSessoes=$(($(echo ${linesA[$user]} | awk '{print $2}') - $(echo ${linesB[$user]} | awk '{print $2}')))
        difTotalTime=$(($(echo ${linesA[$user]}  | awk '{print $3}') - $(echo ${linesB[$user]} | awk '{print $3}')))
        difMaxTime=$(($(echo ${linesA[$user]}  | awk '{print $4}') - $(echo ${linesB[$user]} | awk '{print $4}')))
        difMinTime=$(($(echo ${linesA[$user]}  | awk '{print $5}') - $(echo ${linesB[$user]} | awk '{print $5}')))
        final_info[$user]="$user  $difSessoes $difTotalTime $difMaxTime $difMinTime"
    else
        sessoesA=$(echo ${linesA[$user]} | awk '{print $2}')
        totalTimeA=$(echo ${linesA[$user]} | awk '{print $3}')
        maxTimeA=$(echo ${linesA[$user]} | awk '{print $4}')
        minTimeA=$(echo ${linesA[$user]} | awk '{print $5}')
        final_info[$user]="$user  $sessoesA $totalTimeA $maxTimeA $minTimeA"
    fi
done

for user in "${!linesB[@]}"
do
    if [[ ! ${linesA[$user]} ]]; then
        difSessoes=$((0 - $(echo ${linesB[$user]} | awk '{print $2}')))
        difTotalTime=$((0 - $(echo ${linesB[$user]} | awk '{print $3}')))
        difMaxTime=$((0 - $(echo ${linesB[$user]} | awk '{print $4}')))
        difMinTime=$((0 - $(echo ${linesB[$user]} | awk '{print $5}')))
        final_info[$user]="$user $difSessoes $difTotalTime $difMaxTime $difMinTime"
    fi
done
```

Fig. 12 – Operações entre ficheiros

Para a verificação da presença do utilizador num dos ficheiros ou em ambos, o problema foi feito da seguinte forma: primeiro, com a utilização de um for loop é verificada para todas as **chaves** presentes em **lineA** (utilizadores) através de um **if** a sua presença em **lineB.** Caso isso ocorra, é feita a subtração entre todos os valores de cada utilizador presente tanto em **lineA** como em **lineB**, ou seja, ocorre a subtração das sessões, tempo total de ligação (em minutos), duração máxima e duração mínima das sessões dos utilizadores.
Caso não se verifique a sua presença no **lineB** é impresso apenas os resultados do **lineA.** Toda a informação é guardada num novo array associativo chamado **final_info** que tem como chave o utilizador e como valor todos os resultados das subtrações obtidas.

Do mesmo modo, utilizando um segundo for loop, para cada **user** presente em **lineB** se este não estiver presente no **lineA** é feita a subtração das sessões, tempo total de ligação (em minutos), duração máxima e duração mínima das sessões dos utilizadores, exceto que desta forma irá aparecer como valor negativo para indicar que só ocorre no **lineB.** O resultado, como explicado no primeiro for loop é guardado em **final_info.**

```
order=""
if [[ ${args['r']} ]]; then
    order="r"
fi

if [[ ${args['n']} ]]; then
    printf "%s\n" "${final_info[@]}" | sort -k2,2n$order
elif [[ ${args['t']} ]]; then
    printf "%s\n" "${final_info[@]}" | sort -k3,3n$order
elif [[ ${args['a']} ]]; then
    printf "%s\n" "${final_info[@]}" | sort -k4,4n$order
elif [[ ${args['i']} ]]; then
    printf "%s\n" "${final_info[@]}" | sort -k5,5n$order
else
    printf "%s\n" "${final_info[@]}" | sort -k1,1$order
fi
```

Fig. 13 – Verificação dos argumentos de

Para cada opção introduzida na **bash**, são analisadas as opções presentes no array associativo **args**. Ou seja, caso se verifique a presença da opção **-r** a informação presente no array associativo **final_info** é ordenada de forma decrescente. Do mesmo modo, para a opção **-n** é ordenada por número de seções, para a opção **-t** por tempo total, para a **-a** por tempo máximo e para a **-i** por tempo mínimo.

Testes

Userstats.sh

```
[sop0206@l040101-ws10 Desktop]$ ./userstats.sh
nlau 2 115 107 8
root 2 11 6 5
sd0104 14 762 131 0
sd0105 12 133 64 0
sd0108 6 123 93 0
sd0301 1 0 0 0
sd0303 18 62 60 0
sd0304 1 0 0 0
sd0402 63 0 0 0
sd0406 163 148 131 0
sd0407 2 188 184 4
sop0104 30 18855 11471 0
sop0106 1 0 0 0
sop0206 20 652 135 0
sop0306 15 695 160 0
sop0310 4 661 297 6
sop0402 10 939 199 5
sop0409 2 2 2 0
sop0410 2 20 15 5
```

Fig. 14 – ./userstats.sh

```
rui@rui-VivoBook-S15-X510UF:~/Desktop/SO/Trabalho 1$ ./userstats.sh -f wtmp.1
nlau 7 413 132 5
sop0106 1 0 0 0
sop0110 15 1911 1727 0
sop0202 10 362 137 0
sop0203 2 7 6 1
sop0204 6 188 85 0
sop0206 2 12 10 2
sop0209 5 50 29 0
sop0303 25 1407 117 0
sop0401 3 181 138 6
sop0405 1 2 2 2
sop0406 7 567 283 0
sop0408 4 226 213 0
sop0410 1 10 10 10
sop0411 8 3282 1509 0
sop0413 5 459 112 60
```

Fig. 15 – ./userstats.sh -f wtmp.1

```
[sop0206@l040101-ws10 Desktop]$ ./userstats.sh -u "sop.*"
sop0104 30 18855 11471 0
sop0106 1 0 0 0
sop0206 20 652 135 0
sop0306 15 695 160 0
sop0310 4 661 297 6
sop0402 10 939 199 5
sop0409 2 2 2 0
sop0410 2 20 15 5
```

Fig. 16 – $./userstats.sh -u "sop.*"

```
[sop0206@l040101-ws10 Desktop]$ ./userstats.sh -g sop
sop0104 30 18855 11471 0
sop0106 1 0 0 0
sop0206 20 652 135 0
sop0306 15 695 160 0
sop0310 4 661 297 6
sop0402 10 939 199 5
sop0409 2 2 2 0
sop0410 2 20 15 5
```

Fig. 17 –$./userstats.sh -g sop

```
[sop0206@l040101-ws10 Desktop]$ ./userstats.sh -s "Sep 19 10:00" -e "Sep 23 18:00"
sop0104 30 18855 11471 0
sop0206 20 652 135 0
```

Fig. 18 $ ./userstats.sh -s "Sep 19 10:00" -e "Sep 23 18:00"

```
[sop0206@l040101-ws10 Desktop]$ ./userstats.sh -t -u "sop.*"
sop0102 1 0 0 0
sop0106 1 0 0 0
sop0109 1 0 0 0
sop0409 2 2 2 0
sop0410 2 20 15 5
sop0310 5 661 297 6
sop0306 16 835 160 0
sop0402 10 939 199 5
sop0206 21 961 227 0
sop0104 33 19007 11471 0
```

Fig. 19 – $./userstats.sh -t -u "sop.*"

```
[sop0206@l040101-ws10 Desktop]$ ./userstats.sh -n -u "sop.*"
sop0106 1 0 0 0
sop0409 2 2 2 0
sop0410 2 20 15 5
sop0310 4 661 297 6
sop0402 10 939 199 5
sop0306 15 695 160 0
sop0206 20 652 135 0
sop0104 30 18855 11471 0
```

Fig. 20 – $./userstats.sh -n -u "sop.*"

```
[sop0206@l040101-ws10 Desktop]$ ./userstats.sh -t -r -u "sop.*"
sop0104 30 18855 11471 0
sop0402 10 939 199 5
sop0306 15 695 160 0
sop0310 4 661 297 6
sop0206 20 652 135 0
sop0410 2 20 15 5
sop0409 2 2 2 0
sop0106 1 0 0 0
```

Fig. 21 – $./userstats.sh -t -r -u "sop.*"

```
[sop0206@l040101-ws10 Desktop]$ ./userstats.sh -a -r -u "sop.*"
sop0104 30 18855 11471 0
sop0310 4 661 297 6
sop0402 10 939 199 5
sop0306 15 695 160 0
sop0206 20 652 135 0
sop0410 2 20 15 5
sop0409 2 2 2 0
sop0106 1 0 0 0
```

Fig. 22 – $./userstats.sh -a -r -u "sop.*"

```
[sop0206@l040101-ws10 Desktop]$ ./userstats.sh -i -r -u "sop.*"
sop0310 4 661 297 6
sop0402 10 939 199 5
sop0410 2 20 15 5
sop0104 30 18855 11471 0
sop0106 1 0 0 0
sop0206 20 652 135 0
sop0306 15 695 160 0
sop0409 2 2 2 0
```

Fig. 23 – $./userstats.sh -i -r -u "sop.*"

```
[sop0206@l040101-ws10 Desktop]$ ./userstats.sh -u ".*" > userstats_123
[sop0206@l040101-ws10 Desktop]$ nano userstats_123
```

```
nlau 2 115 107 8
root 2 11 6 5
sd0104 14 762 131 0
sd0105 12 133 64 0
sd0108 6 123 93 0
sd0301 1 0 0 0
sd0303 18 62 60 0
sd0304 1 0 0 0
sd0402 63 0 0 0
sd0406 163 148 131 0
sd0407 2 188 184 4
sop0104 30 18855 11471 0
sop0106 1 0 0 0
sop0206 20 652 135 0
sop0306 15 695 160 0
sop0310 4 661 297 6
sop0402 10 939 199 5
sop0409 2 2 2 0
sop0410 2 20 15 5
```

Fig. 24 – $./userstats.sh -n ".*" > userstats_123

Comparestats.sh

```
alex@sop0206:~/Desktop/SO trabalho$ ./comparestats.sh test1.txt test.txt
nlau  -6 -547 -144 0
root  2 11 6 5
sd0104  7 38 -99 -18
sd0105  2 26 20 0
sd0106 -6 -139 -73 -1
sd0108  6 123 93 0
sd0109 -1 -2 -2 -2
sd0301  -59 0 0 0
sd0302 -125 -720 -131 0
sd0303  -10 58 58 0
sd0304  0 0 0 0
sd0305 -20 -1 -1 0
sd0401 -21 0 0 0
sd0402  -26 -131 -131 0
sd0403 -1 -8 -8 -8
sd0405 -610 -45 -10 0
sd0406  -16 -184 -1 0
sd0407  1 1 -3 -183
sop0104  25 18591 11471 0
sop0106  0 0 0 0
sop0109 -18 -2163 -1317 0
sop0205 -48 -9090 -6366 0
sop0206  18 652 135 0
sop0207 -6 -16 -6 0
sop0304 -11 -17346 -8371 0
sop0402  2 174 88 86
sop0405 -8 -723 -99 -69
sop0407 -2 -15 -15 0
sop0410 -73 -4283 -191 0
```

Fig. 25 – $./comparestats.sh test1.txt test.txt

```
alex@sop0206:~/Desktop/SO trabalho$ ./comparestats.sh test.txt test1.txt
nlau  6 547 144 0
root -2 -11 -6 -5
sd0104  -7 -38 99 18
sd0105  -2 -26 -20 0
sd0106  6 139 73 1
sd0108 -6 -123 -93 0
sd0109  1 2 2 2
sd0301  59 0 0 0
sd0302  125 720 131 0
sd0303  10 -58 -58 0
sd0304  0 0 0 0
sd0305  20 1 1 0
sd0401  21 0 0 0
sd0402  26 131 131 0
sd0403  1 8 8 8
sd0405  610 45 10 0
sd0406  16 184 1 0
sd0407  -1 -1 3 183
sop0104 -25 -18591 -11471 0
sop0106  0 0 0 0
sop0109  18 2163 1317 0
sop0205  48 9090 6366 0
sop0206 -18 -652 -135 0
sop0207  6 16 6 0
sop0304  11 17346 8371 0
sop0402 -2 -174 -88 -86
sop0405  8 723 99 69
sop0407  2 15 15 0
sop0410  73 4283 191 0
```

Fig. 26 – $./comparestats.sh test.txt test1.txt

```
alex@sop0206:~/Desktop/SO trabalho$ ./comparestats.sh test1.txt test1.txt
nlau  0 0 0 0
root  0 0 0 0
sd0104  0 0 0 0
sd0105  0 0 0 0
sd0108  0 0 0 0
sd0301  0 0 0 0
sd0303  0 0 0 0
sd0304  0 0 0 0
sd0402  0 0 0 0
sd0406  0 0 0 0
sd0407  0 0 0 0
sop0104  0 0 0 0
sop0106  0 0 0 0
sop0206  0 0 0 0
sop0402  0 0 0 0
```

Fig. 27 – $./comparestats.sh test1.txt test1.txt

```
alex@sop0206:~/Desktop/SO trabalho$ ./comparestats.sh  -a test.txt test1.txt
sop0104 -25 -18591 -11471 0
sop0206 -18 -652 -135 0
sd0108 -6 -123 -93 0
sop0402 -2 -174 -88 -86
sd0303  10 -58 -58 0
sd0105  -2 -26 -20 0
```

Fig. 28 – $./comparestats.sh -a test.txt test1.txt

```
alex@sop0206:~/Desktop/SO trabalho$ ./comparestats.sh  -i test.txt test1.txt
sop0402 -2 -174 -88 -86
root -2 -11 -6 -5
nlau  6 547 144 0
sd0105  -2 -26 -20 0
sd0108 -6 -123 -93 0
sd0301  59 0 0 0
sd0302  125 720 131 0
sd0303  10 -58 -58 0
```

Fig. 29 – $./comparestats.sh -i test.txt test1.txt

```
alex@sop0206:~/Desktop/SO trabalho$ ./comparestats.sh -n test.txt test1.txt
sop0104 -25 -18591 -11471 0
sop0206 -18 -652 -135 0
sd0104  -7 -38 99 18
sd0108 -6 -123 -93 0
root -2 -11 -6 -5
sd0105  -2 -26 -20 0
sop0402 -2 -174 -88 -86
sd0407  -1 -1 3 183
sd0304  0 0 0 0
sop0106  0 0 0 0
```

Fig. 30 – $./comparestats.sh -n test.txt test1.txt

```
lex@sop0206:~/Desktop/SO trabalho$ ./comparestats.sh  -t test.txt test1.txt
op0104 -25 -18591 -11471 0
op0206 -18 -652 -135 0
op0402 -2 -174 -88 -86
d0108 -6 -123 -93 0
d0303  10 -58 -58 0
```

Fig. 31 – $./comparestats.sh -t test.txt test1.txt

Testes de Erros

```
[sop0206@l040101-ws10 Desktop]$ ./userstats.sh -t -n 123
ERROR: too many ordering options, use only one of the following: -n -t -a -i
```

Fig. 32 – Too many ordering options

```
sop0206@l040101-ws10 Desktop]$ ./userstats.sh abcd
RROR: parameters are not valid: abcd
```

Fig. 33 ERRO: parameters are not valid: ___

```
[sop0206@l040101-ws10 Desktop]$ ./userstats.sh -f fileRrror
ERROR: file not found!
```

Fig. 34 – ERROR: file not found!

```
rui@rui-VivoBook-S15-X510UF:~/Desktop/SO/Trabalho 1$ ./userstats.sh -x
ERROR: invalid option: -x
```

Fig. 35 – -x invalid option

```
rui@rui-VivoBook-S15-X510UF:~/Desktop/SO/Trabalho 1$ ./userstats.sh -g
ERROR: option -g needs an argument.
```

Fig. 36 – option needs argument

```
rui@rui-VivoBook-S15-X510UF:~/Desktop/SO/Trabalho 1$ ./comparestats.sh -n -a test.txt test1.txt
ERROR: too many ordering options, use only one of the following: -n -t -a -i
```

Fig. 37 – Too many ordering options

```
rui@rui-VivoBook-S15-X510UF:~/Desktop/SO/Trabalho 1$ ./comparestats.sh
ERROR: The program requires exactly 2 files!
rui@rui-VivoBook-S15-X510UF:~/Desktop/SO/Trabalho 1$ ./comparestats.sh test.txt test1.txt test2.txt
ERROR: The program requires exactly 2 files!
```

Fig. 38 – Wrong file amount

```
rui@rui-VivoBook-S15-X510UF:~/Desktop/SO/Trabalho 1$ ./comparestats.sh test.txt testerr.txt
ERROR: file testerr.txt not found!
```

Fig. 39 – File not found

Conclusão

Tanto o userstats.sh como comparestats.sh foram implementados com êxito, ou seja, todos os resultados estão em conformidade com o pedido e apresentado nos exemplos do enunciado.

Este trabalho foi feito com base em muita pesquisa, sendo que grande parte do conhecimento foi adquirido nas aulas teóricas e práticas previamente. Do modo geral, o maior desafio foi perceber todos os comandos necessários para a implementação do projeto a organização das ideias, visto que todo o código teria que ter uma estrutura e um funcionamento apropriado conforme o enunciado do projeto.

Finalmente, todos os testes realizados foram bem sucedidos.

Bibliografia

https://devhints.io/bash

https://www.artificialworlds.net/blog/2012/10/17/bash-associative-array-examples/

https://www.geeksforgeeks.org/last-command-in-linux-with-examples/

https://pt.stackoverflow.com/