Trabalho Prático 1

Sistemas Operativos

Bruno Bastos 93302

Eduardo Santos 93107

Índice

Introdução

O primeiro trabalho prático de sistemas operativos tem por base o uso do comando “last”, que nos dá informação sobre as últimas sessões que foram feitas nesse computador. Tem como objetivo a realização de dois programas, “userstats” e “comparestats”, onde no primeiro será calculada a informação sobre cada utilizador e será impressa no terminal, enquanto que o segundo calcula as diferenças entre as informações dos utilizadores previamente calculadas no primeiro. Ambos os programas dispõem de várias opções de ordenação e procura.

Explicação das Opções

O programa “userstats” suporta várias opções que fazem uso do comando “last” para calcular e mostrar a informação relativa aos utilizadores.

Opção: -u “regex”

A opção -u requer um argumento obrigatório, sendo este uma expressão regular para filtrar os utilizadores por nome. Caso não existam utilizadores que satisfaçam a expressão regular, o programa retorna uma mensagem a dizer que não foram encontrados utilizadores.

Opção: -g “group”

A opção -g requer um argumento obrigatório que corresponde ao grupo pelo qual se quer filtrar os utilizadores. Caso nenhum dos utilizadores pertença ao grupo introduzido o programa retorna uma mensagem a dizer que não foram encontrados utilizadores.

Opção: -s “Date”

A opção -s com um argumento obrigatório no formato “date”, permite ao utilizador especificar a partir de quando os dados sobres os utilizadores devem ser avaliados. Caso o argumento introduzido não seja do tipo “date” o programa retornará uma mensagem de erro.

Opção: -e “Date”

Assim como a opção -s, a opção -e requer um argumento obrigatório no formato “date”. Esta opção permite ao utilizador especificar o momento a partir do qual não devem ser contados as sessões dos utilizadores. À imagem da opção -s, esta opção também retorna uma mensagem de erro caso não seja inserida uma data válida.

Opção: -f “File”

A opção -f requer um ficheiro como argumento. O comando “last” irá obter as informações sobre as sessões a partir desse ficheiro. Caso o ficheiro não exista o programa retorna uma mensagem de erro.

Opção: -r

A opção -r não tem argumentos. Esta opção serve para ordenar por ordem reversa o output do programa.

Opção: -n

A opção -n não tem argumentos e não pode ser usada em conjunto com outras opções de ordenação. A ordenação vai ser feita a partir do número de sessões de cada utilizador de forma ascendente. É possível utilizar a opção -r para alterar a ordem desta ordenação.

Opção: -t

A opção -t não tem argumentos e não pode ser usada em conjunto com outras opções de ordenação. A ordenação terá como chave de ordenação o número total de horas de cada utilizador. A opção -r altera a ordem com que os resultados serão mostrados no ecrã.

Opção: -a

A opção -a não tem argumentos e não pode ser usada em conjunto com outras opções de ordenação. A ordenação tem por base o número máximo de horas de cada utilizador em uma só sessão e assim como as outras opções de ordenação, também esta pode ser usada em conjunto com a opção -r para inverter a ordem dos resultados.

Opção: -i

A opção -i não tem argumentos e não pode ser usada em conjunto com as outras opções de ordenação. A ordenação é feita tendo em conta o número de mínimo de horas de cada utilizador numa só sessão. A opção -r inverte a ordem dos resultados.

O programa “userstats” faz uso de todas estas opções, enquanto que o program “comparestats” apenas usa as opções de ordenação dos resultados.

Explicação do Código

Userstats.sh

No começo do ficheiro são definidas 3 variáveis, “sort” que vai servir como chave de ordenação, “sorting” que vai prevenir que hajam várias chaves de ordenação e “last” que vai ser o comando usado para ir buscar a informação das sessões.

```
sort="sort";
sorting="";
last="last -w"
```

Foi usada a opção -w no “last” pois esta devolve os nomes completos dos usuários que vai ajudar para verificar o grupo de cada utilizador.

Segue-se a função “getopts” que vai verificar quais as opções introduzidas pelo o utilizador e efetuar o comando respetivo.

```
while getopts "::g:u:s:e:f::nrtai" o; do
    case "${o}" in
        u)
            u=${OPTARG}
            ;;
        f)
            f=${OPTARG}
            last="$last -f $f"
            ;;
        n)
            if [[ $sorting = "s" ]];then
                echo "ERROR : More than 2 sorting options were selected"
                exit 1;
```

```
            fi
            sort="$sort -k2 -n ";
            sorting="s";
            ;;
        r)
            sort="$sort -r";
            ;;
        t)
            if [[ $sorting = "s" ]];then
                echo "ERROR : More than 2 sorting options were selected"
                exit 1;
            fi
            sort="$sort -n -k3 ";
            sorting="s";
            ;;
        a)
            if [[ $sorting = "s" ]];then
                echo "ERROR : More than 2 sorting options were selected"
                exit 1;
            fi
            sort="$sort -n -k4 ";
            sorting="s";
            ;;
        i)
            if [[ $sorting = "s" ]];then
                echo "ERROR : More than 2 sorting options were selected"
                exit 1;
            fi
            sort="$sort -n -k5 ";
            sorting="s";
            ;;
        g)
            group=${OPTARG};
            ;;
        s)
            start_date=$(date -d"$OPTARG" "+%y-%m-%d %H:%M" );
            last="$last -s \"$start_date\"";
            ;;
        e)
            end_date=$(date -d "$OPTARG" "+%Y-%m-%d %H:%M");
            last="$last -t \"$end_date\"";
            ;;
        *)
            getRules
            exit 1
            ;;
    esac
done
shift $((OPTIND-1))
```

São feitas verificações para que não sejam usadas 2 argumentos de ordenação. Em alguns casos as variaveis “sort” e “last” são alteradas de modo a executarem o que é pretendido. No caso das opções -s e -e, os argumentos são convertidos para o formato “ano-mes-dia hora:min”. Caso sejam introduzidas opções não pretendidas, o programa dá output a uma tabela de ajuda com as opções existentes.

De seguida é feita uma verificação sobre a existência do ficheiro introduzido na opção -f caso esta seja usada.

```
if [[ -z $f ]];then
    f="/var/log/wtmp"
else
    if ! [[ -f $f ]];then
        echo "ERROR : Can not read from file."
        exit 1
    fi
fi
```

É verificada a existência da variável “u”, que corresponde ao argumento da opção -u. Caso este exista, é usado o comando last com a informação presente na variável “last”, seguido de um “awk” para olhar apenas para a primeira coluna e depois é realizado um “grep” para escolher apenas aqueles que correspondam à expressão regular passada como argumento. Se a variável “u” não existir o processo é o mesmo, porém não será usado o comando “grep”. É de salientar que serão retirados quaisquer elementos da coluna 1 que não correspondam a nomes de utilizadores.

```
if [[ -n "$u" ]];then
        users=($(eval $last | grep -v "logged" | awk '{print $1}' | grep -v "reboot" | grep -
v "wtmp" | sort |  uniq | grep "$u" ))
    else
        users=($(eval $last | grep -v "logged" | awk '{print $1}' | grep -v "reboot" | grep -
v "wtmp" | sort | uniq | tr '\n' ' '))
     fi
```

Depois será feita a verificação da existência da variável "group" que corresponde ao argumento passado na utilização da opção -g. Se isto se verificar será percorrido o array com os nomes dos utilizadores e será feita a comparação entre a variável "group" e os groups de cada utilizador. Aqueles que não satisfizerem esta condição serão removidos do array "users".

```
if [[ -n "$group" ]];then
        for g in ${users[@]}
        do
            isGroup="n"
            usersgroup=($(groups "$g"));
            for i in ${usersgroup[@]}
            do
                if [[ "$i" = "$group" ]];then
                    isGroup="y"
                    break
                fi
            done

            if [[ "$isGroup" = "n" ]];then
                users=(${users[@]/$g})
            fi

        done
    fi
```

Com o array “users” filtrado serão percorridos todos os seus elementos. Para cada um serão contadas as vezes em que este aparece no comando last usando o comando “wc -l” e serão também somadas os tempos de cada sessão. Como cada utilizador poderá ter mais de uma sessão, os tempos são guardados num array que irá ser percorrido de seguida. São removidas algumas sessões que não têm os tempos. É usado o primeiro valor presente no array para que este inicialize as variáveis do tempo máximo e do tempo mínimo. Como o tamanho da string dos tempos pode variar devido ao facto de haver sessões que tem mais de um dia, será feita uma verificação para que não hajam erros no cálculo dos tempos de sessão.

```
if  [[ ${users[@]} ]] ;then
    for i in "${users[@]}"
    do

        counter=$(eval $last  | awk '{print $1;}' | grep "$i" | wc -l )
        time=($(eval $last  | grep "$i" | awk '{print $10}' | grep -v "in" | grep -
v "no" | grep -v "on" | grep -v "logged" | grep -v "running" | grep -v "still"))

        y=${time[0]};
        if [[ ${#y} -gt 7 ]];then
            day=$(echo $y | cut -d '+' -f 1 | cut -d '(' -f 2);
            hour=$(echo $y | cut -d '+' -f 2 | cut -d ')' -f 1 | cut -d ':' -
f 1 | awk '{sub(/^0/,"");}1');
            min=$(echo $y | cut -d '+' -f 2 | cut -d ')' -f 1 | cut -d ':' -
f 2 | awk '{sub(/^0/,"");}1');
            min_time=$(($day*24*60+$hour*60 + $min))
            max_time=$(($day*24*60+$hour*60 + $min))
        else
            hour=$(echo $y | cut -d '(' -f 2 | cut -d ':' -f 1 | awk '{sub(/^0/,"");}1');
            min=$(echo $y | cut -d '+' -f 2 | cut -d ')' -f 1 | cut -d ':' -
f 2 | awk '{sub(/^0/,"");}1');
            min_time=$(($hour*60 + $min))
            max_time=$(($hour*60 + $min))
        fi
```

Agora será percorrido o array com os tempos do utilizador e será calculado o tempo total somando todos os tempos de cada utilizador e também serão obtidos o tempo de sessão mínimo e o tempo de sessão máxima.

```
for k in "${time[@]}"
        do

            if [[ ${#k} -gt 8 ]];then
                day=$(echo $k | cut -d '+' -f 1 | cut -d '(' -f 2);
                hour=$(echo $k | cut -d '+' -f 2 | cut -d ')' -f 1 | cut -d ':' -
f 1 | awk '{sub(/^0/,"");}1');
                min=$(echo $k | cut -d '+' -f 2 | cut -d ')' -f 1 | cut -d ':' -
f 2 | awk '{sub(/^0/,"");}1');
                min_time=$(($day*24*60+$hour*60 + $min))
                max_time=$(($day*24*60+$hour*60 + $min))
                temp_time=$(( $day*24*60+$hour*60 + $min ))
            else
                hour=$(echo $y | cut -d '+' -f 2 | cut -d '(' -f 2 | cut -d ':' -
f 1 | awk '{sub(/^0/,"");}1');
                min=$(echo $y | cut -d '+' -f 2 | cut -d ')' -f 1 | cut -d ':' -
f 2 | awk '{sub(/^0/,"");}1');
                temp_time=$(($hour*60 + $min))
            fi

            if [[ $temp_time -lt $min_time ]];then
                min_time=$temp_time;
            fi

            if [[ $temp_time -gt $max_time ]];then
                max_time=$temp_time;
            fi
            total_time=$(( $total_time+$temp_time ))
        done
```

No final, a informação de cada utilizador é adicionada ao array userstats.

```
userstats[contador]=$(echo "$i $counter $total_time $max_time $min_time")
        let "contador=contador +1";
```

Para terminar o array é ordenado com base na variável “sort” e é impresso o resultado no ecrã.

```
IFS=$' '
printf '%s \n' "${userstats[@]}" | $sort
```

CompareStats

O programa ComapareStats dá uso às mesmas opções de ordenação usadas no programa anterior. São também passados como argumentos 2 ficheiros com a informação que é obtida através do programa userstats.

No início é feita uma verificação sobre a existência dos ficheiros passados como argumentos. Caso existam, ambos os ficheiros são lidos e o seu conteúdo é guardado em um array.

```
if ! [[ -f $1 ]];then
        echo "cannot read from file"
        exit 1
    fi
    if ! [[ -f $2 ]];then
        echo "cannot read from file"
        exit 1
    fi
    while IFS= read -r line; do
        IFS=$'\n'
        users1+=($line);
    done < $1
    while IFS= read -r line; do
        IFS=$'\n'
        users2+=($line);
    done < $2
```

Depois são percorridos os dois arrays ao mesmo tempo e procura-se por elementos dos arrays que tenham o mesmo nome. Será então calculada a diferença entre as suas informações, tendo em conta que o primeiro ficheiro é o mais recente. Depois de calculadas, as informações serão guardadas em um array.

```
for k in ${users1[@]}
    do
        name1=$(echo "$k" | awk '{print $1}');
        for i in ${users2[@]}
        do
            name2=$(echo "$i" | awk '{print $1}');
            if [[ "$name1" = "$name2" ]] ; then
                let "sessions=$(( $(echo "$k" | awk '{print $2}') - $(ech
o "$i" | awk '{print $2}') ))";
                let "time=$(( $(echo "$k" | awk '{print $3}') - $(echo "$
i" | awk '{print $3}') ))"
                let "max_time=$(( $(echo "$k" | awk '{print $4}') - $(ech
o "$i" | awk '{print $4}') ))"
                let "min_time=$(( $(echo "$k" | awk '{print $5}') - $(ech
o "$i" | awk '{print $5}') ))"
                userstats[counter]=$(echo "$name1 $sessions $time $max_ti
me $min_time ");
                let "counter=counter +1"
            fi

        done

    done
```

De seguida serão adicionados ao array os elementos não comuns dos ficheiros. Por fim, as informações são transpostas para o ecrã.

```
for i in ${users1[@]}
do
    username=$(echo "$i" | awk '{print $1}');
    add="y"
    for j in ${userstats[@]}
    do
        temp_name=$(echo "$j" | awk '{print $1}');
        if [[ "$username" = "$temp_name" ]];then
            add="n"
        fi
    done
    if [[ "$add" = "y" ]];then
        userstats[counter]=$(echo "$i")
        let "counter=counter+1"
    fi
done
for i in ${users2[@]}
do
    add="y"
    username=$(echo "$i" | awk '{print $1}');
    for j in ${userstats[@]}
    do
        temp_name=$(echo "$j" | awk '{print $1}');
        if [[ "$username" = "$temp_name" ]];then
            add="n"
        fi
    done

    if [[ "$add" = "y" ]];then
        userstats[counter]=$(echo "$i")
        let "counter=counter+1"
    fi
done

IFS=$' '
printf '%s \n' "${userstats[@]}" | $sort
```

Testes

Segue-se agora os testes a ambos os ficheiros.

Userstats.sh

Sem opções:

```
[sop0308@l040101-ws04 ~]$ bash ./userstats.sh
sop0101 2 116 58 58
sop0102 1 0 0 0
sop0308 3 20 10 10
sop0404 14 0 0 0
sop0405 4 528 132 132
sop0406 10 20 2 2
sop0407 6 54 9 9
sop0409 1 2 2 2
```

Opção -r:

```
[sop0308@l040101-ws04 ~]$ bash ./userstats.sh -r
sop0409 1 2 2 2
sop0407 6 54 9 9
sop0406 10 20 2 2
sop0405 4 528 132 132
sop0404 14 0 0 0
sop0308 3 20 10 10
sop0102 1 0 0 0
sop0101 2 116 58 58
```

Opção -u:

```
[sop0308@l040101-ws04 ~]$ bash ./userstats.sh -u "sop0101"
sop0101 2 116 58 58
[sop0308@l040101-ws04 ~]$ _
```

```
[sop0308@l040101-ws04 ~]$ bash ./userstats.sh -u "nlau"
No users found.
```

Opção -g:

```
[sop0308@l040101-ws04 ~]$ bash ./userstats.sh -g "sop"
sop0101 2 116 58 58
sop0102 1 0 0 0
sop0308 3 20 10 10
sop0404 14 0 0 0
sop0405 4 528 132 132
sop0406 10 20 2 2
sop0407 6 54 9 9
sop0409 1 2 2 2
```

```
[sop0308@l040101-ws04 ~]$ bash ./userstats.sh -g "sudo"
No users found.
```

Opção -s:

```
[sop0308@l040101-ws04 ~]$ bash ./userstats.sh -s "Nov 22 23:02"
sop0101 2 116 58 58
sop0102 1 0 0 0
sop0308 3 20 10 10
sop0404 7 0 0 0
sop0405 3 396 132 132
sop0406 9 18 2 2
sop0407 6 54 9 9
```

Opção -e:

```
[sop0308@l040101-ws04 ~]$ bash ./userstats.sh -e "Nov 22 00:00"
sop0404 4 1372 343 343
sop0406 1 53 53 53
sop0409 1 2 2 2
```

Opção -s e -e:

```
[sop0308@l040101-ws04 ~]$ bash ./userstats.sh -e "Nov 22 00:00" -s "Nov 21 11:00"
sop0409 1 2 2 2
```

Opção -f:

```
brunobastos@brunobastos-HP-Pavilion-Laptop-15-ck0xx:~/Desktop$ ./userstats.sh -f wtmp
nlau 6 0 0 0
sd0104 4 920 230 230
sd0105 10 50 5 5
sd0106 1 0 0 0
sd0109 2 26 13 13
sd0301 60 0 0 0
sd0302 256 256 1 1
sd0303 28 0 0 0
sd0304 1 0 0 0
sd0305 20 0 0 0
sd0401 21 0 0 0
sd0402 90 0 0 0
sd0403 1 9 9 9
sd0405 610 0 0 0
sd0406 182 2184 12 12
sd0407 2 272 136 136
sop0101 14 1428 102 102
sop0106 1 0 0 0
sop0202 17 6042 4410 102
sop0301 3 17111 8486 8486
sop0402 18 1224 68 68
sop0406 10 9 1 1
```

Opção -n:

```
[sop0308@l040101-ws04 ~]$ bash ./userstats.sh -n
sop0102 1 0 0 0
sop0409 1 2 2 2
sop0101 2 116 58 58
sop0308 3 20 10 10
sop0405 4 528 132 132
sop0407 6 54 9 9
sop0406 10 20 2 2
sop0404 14 0 0 0
```

```
[sop0308@l040101-ws04 ~]$ bash ./userstats.sh -n -r
sop0404 14 0 0 0
sop0406 10 20 2 2
sop0407 6 54 9 9
sop0405 4 528 132 132
sop0308 3 20 10 10
sop0101 2 116 58 58
sop0409 1 2 2 2
sop0102 1 0 0 0
```

Opção -t:

```
[sop0308@l040101-ws04 ~]$ bash ./userstats.sh -t
sop0102 1 0 0 0
sop0404 14 0 0 0
sop0409 1 2 2 2
sop0308 3 20 10 10
sop0406 10 20 2 2
sop0407 6 54 9 9
sop0101 2 116 58 58
sop0405 4 528 132 132
```

```
[sop0308@l040101-ws04 ~]$ bash ./userstats.sh -t -r
sop0405 4 528 132 132
sop0101 2 116 58 58
sop0407 6 54 9 9
sop0406 10 20 2 2
sop0308 3 20 10 10
sop0409 1 2 2 2
sop0404 14 0 0 0
sop0102 1 0 0 0
```

Opção -i:

```
[sop0308@l040101-ws04 ~]$ bash ./userstats.sh -i
sop0102 1 0 0 0
sop0404 14 0 0 0
sop0406 10 20 2 2
sop0409 1 2 2 2
sop0407 6 54 9 9
sop0308 3 20 10 10
sop0101 2 116 58 58
sop0405 4 528 132 132
```

```
[sop0308@l040101-ws04 ~]$ bash ./userstats.sh -i -r
sop0405 4 528 132 132
sop0101 2 116 58 58
sop0308 3 20 10 10
sop0407 6 54 9 9
sop0409 1 2 2 2
sop0406 10 20 2 2
sop0404 14 0 0 0
sop0102 1 0 0 0
```

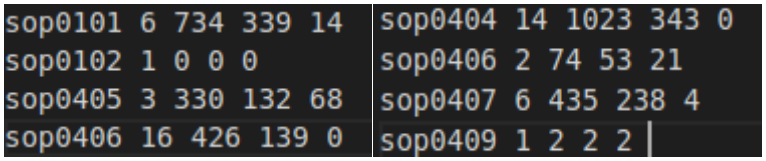
Opção -a:

```
[sop0308@l040101-ws04 ~]$ bash ./userstats.sh -a
sop0102 1 0 0 0
sop0404 14 0 0 0
sop0406 10 20 2 2
sop0409 1 2 2 2
sop0407 6 54 9 9
sop0308 3 20 10 10
sop0101 2 116 58 58
sop0405 4 528 132 132
```

```
[sop0308@l040101-ws04 ~]$ bash ./userstats.sh -a -r
sop0405 4 528 132 132
sop0101 2 116 58 58
sop0308 3 20 10 10
sop0407 6 54 9 9
sop0409 1 2 2 2
sop0406 10 20 2 2
sop0404 14 0 0 0
sop0102 1 0 0 0
```

CompareStats

Para os testes do programa CompareStats foram usados 2 ficheiros:



Recent Old

```
brunobastos@brunobastos-HP-Pavilion-Laptop-15-ck0xx:~/Desktop$ bash compare_stats.sh recent old
sop0101 6 734 339 14
sop0102 1 0 0 0
sop0404 14 1023 343 0
sop0405 3 330 132 68
sop0406 14 352 86 -21
sop0407 6 435 238 4
sop0409 1 2 2 2
brunobastos@brunobastos-HP-Pavilion-Laptop-15-ck0xx:~/Desktop$ bash compare_stats.sh -n recent old
sop0102 1 0 0 0
sop0409 1 2 2 2
sop0405 3 330 132 68
sop0101 6 734 339 14
sop0407 6 435 238 4
sop0404 14 1023 343 0
sop0406 14 352 86 -21
brunobastos@brunobastos-HP-Pavilion-Laptop-15-ck0xx:~/Desktop$ bash compare_stats.sh -r recent old
sop0409 1 2 2 2
sop0407 6 435 238 4
sop0406 14 352 86 -21
sop0405 3 330 132 68
sop0404 14 1023 343 0
sop0102 1 0 0 0
sop0101 6 734 339 14
brunobastos@brunobastos-HP-Pavilion-Laptop-15-ck0xx:~/Desktop$ bash compare_stats.sh -t recent old
sop0102 1 0 0 0
sop0409 1 2 2 2
sop0405 3 330 132 68
sop0406 14 352 86 -21
sop0407 6 435 238 4
sop0101 6 734 339 14
sop0404 14 1023 343 0
brunobastos@brunobastos-HP-Pavilion-Laptop-15-ck0xx:~/Desktop$ bash compare_stats.sh -a recent old
sop0102 1 0 0 0
sop0409 1 2 2 2
sop0406 14 352 86 -21
sop0405 3 330 132 68
sop0407 6 435 238 4
sop0101 6 734 339 14
sop0404 14 1023 343 0
brunobastos@brunobastos-HP-Pavilion-Laptop-15-ck0xx:~/Desktop$ bash compare_stats.sh -i recent old
sop0406 14 352 86 -21
sop0102 1 0 0 0
sop0404 14 1023 343 0
sop0409 1 2 2 2
sop0407 6 435 238 4
sop0101 6 734 339 14
sop0405 3 330 132 68
```

Conclusão

A realização deste trabalho prático deu-nos a conhecer melhor o funcionamento da bash. A bash é um interpretador de comandos que nos dá um enorme controlo sobre o sistema operativo e, portanto, é bom termos uma noção daquilo que ela pode fazer.

Bibliografia

https://www.howtogeek.com/howto/ubuntu/see-which-groups-your-linux-user-belongs-to/

https://unix.stackexchange.com/questions/23111/what-is-the-eval-command-in-bash

https://stackoverflow.com/questions/19482123/extract-part-of-a-string-using-bash-cut-split/19482947

https://stackoverflow.com/questions/16860877/remove-an-element-from-a-bash-array