

Sistemas Operativos

Prof. Nuno Lau



Alexandre Rodrigues - 92951  
Rui Fernandes - 92952

**Estatísticas de utilizadores em bash**

A participação é repartida de igual forma.

**2019/2020**

**Índice**

[Introdução 2](#_Toc25880052)

[Metodologia usada para Solucionar o Problema Userstats.sh 3](#_Toc25880053)

[Metodologia usada para Solucionar o Problema Comparestats.sh 7](#_Toc25880054)

[Testes 10](#_Toc25880055)

[Testes de Erros 13](#_Toc25880064)

[Conclusão 14](#_Toc25880065)

[Bibliografia 15](#_Toc25880066)

# 

# 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 documento irão ser apresentadas todas as etapas e respetivas descrições sobre a metodologia usada para o desenvolvimento do projeto.

# **Metodologia usada para Solucionar o Problema**

**Userstats.sh**

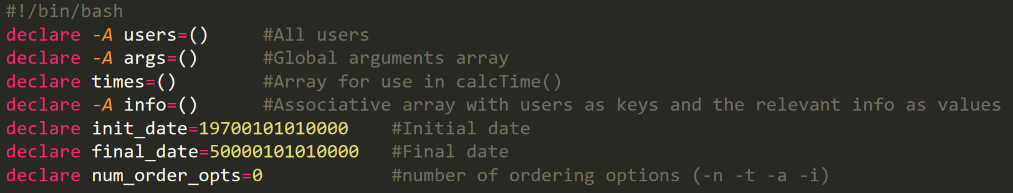


Fig. 1 – Declaração de arrays e variáveis.

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**

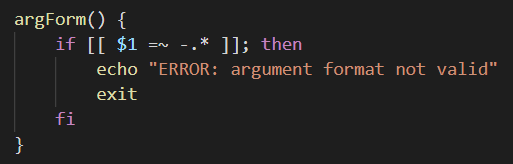
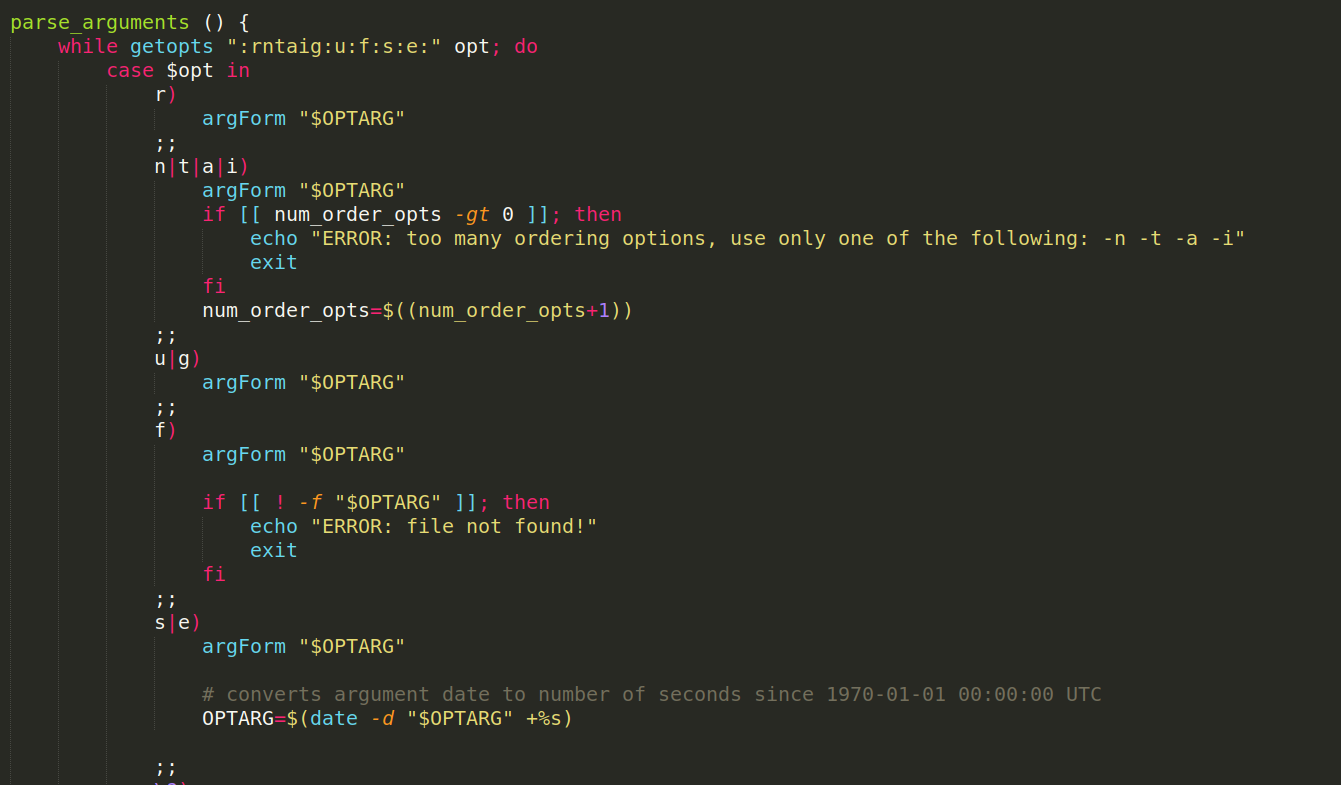


Fig. 2 – Função *argForm()*

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.



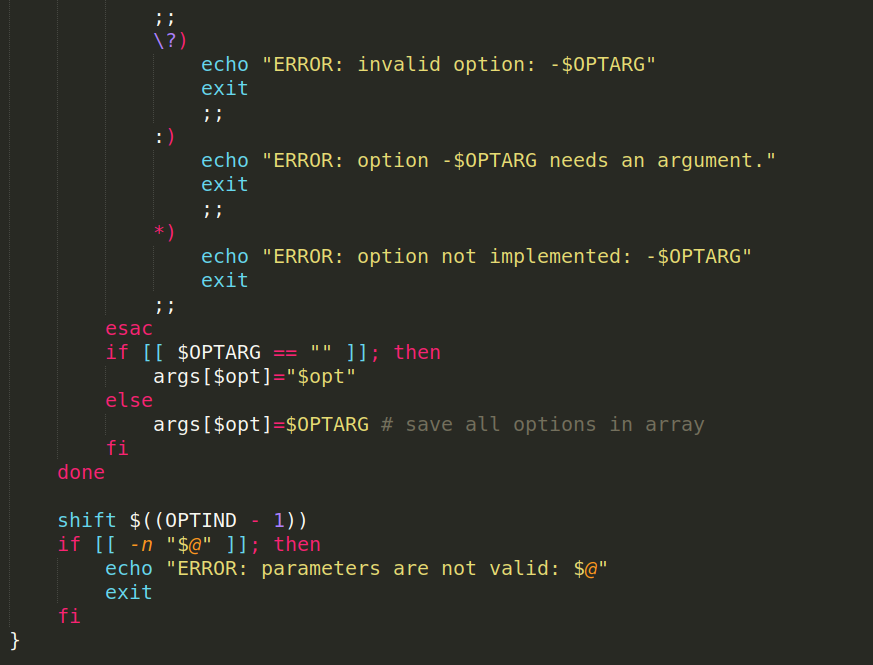


Fig. 3 – Função *parse\_arguments()*

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()**

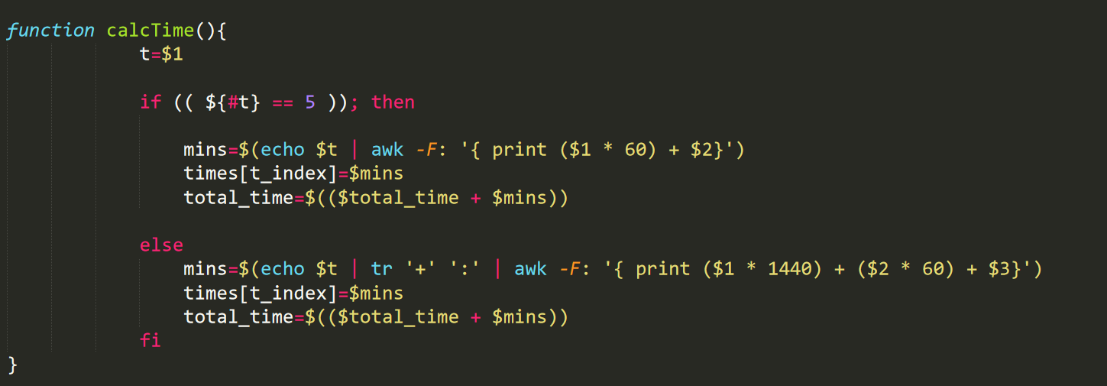


Fig. 4 – Função *calcTime()*

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**

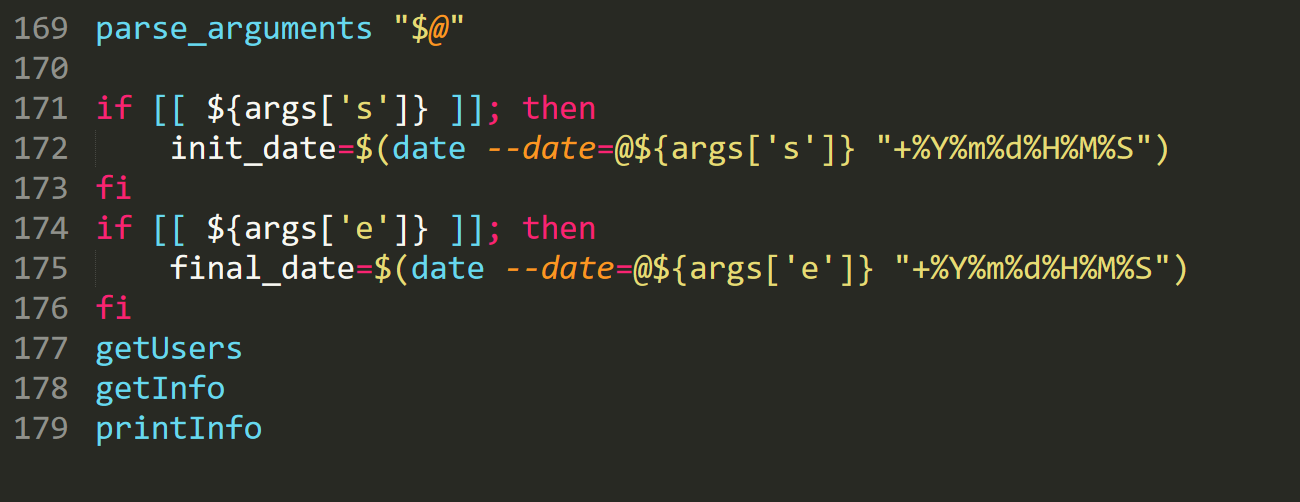
**Opções *-s* e *-e***

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**

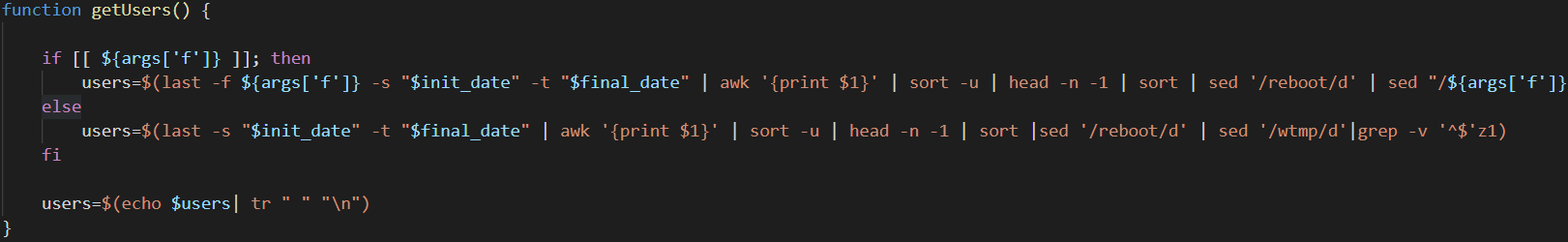


Fig. 6 – Função *getUsers()*

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**

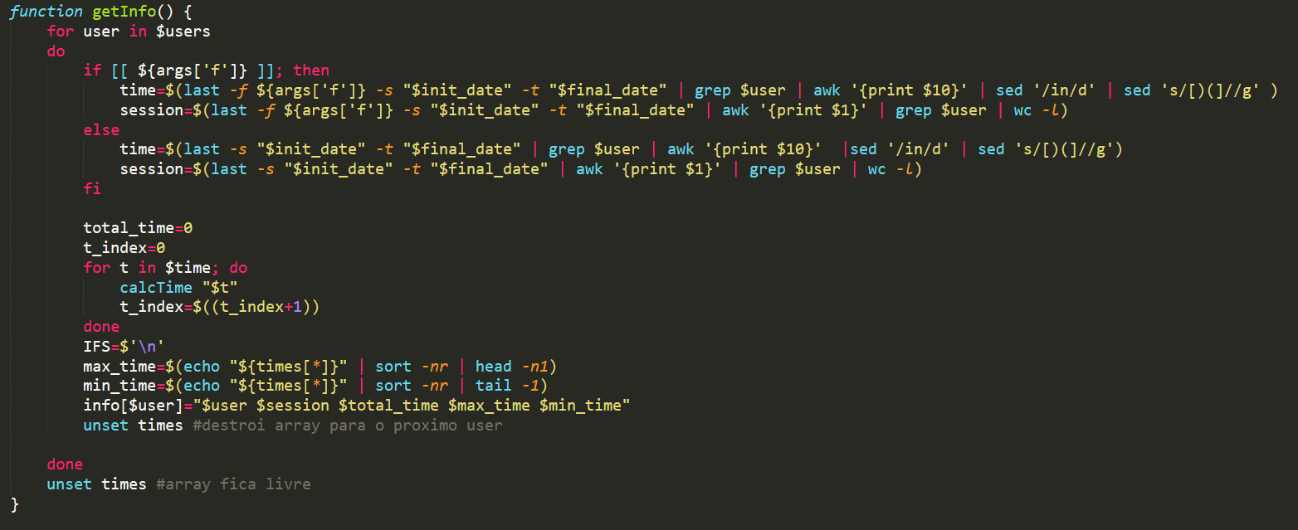


Fig. 7 – Função *getInfo()*

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.

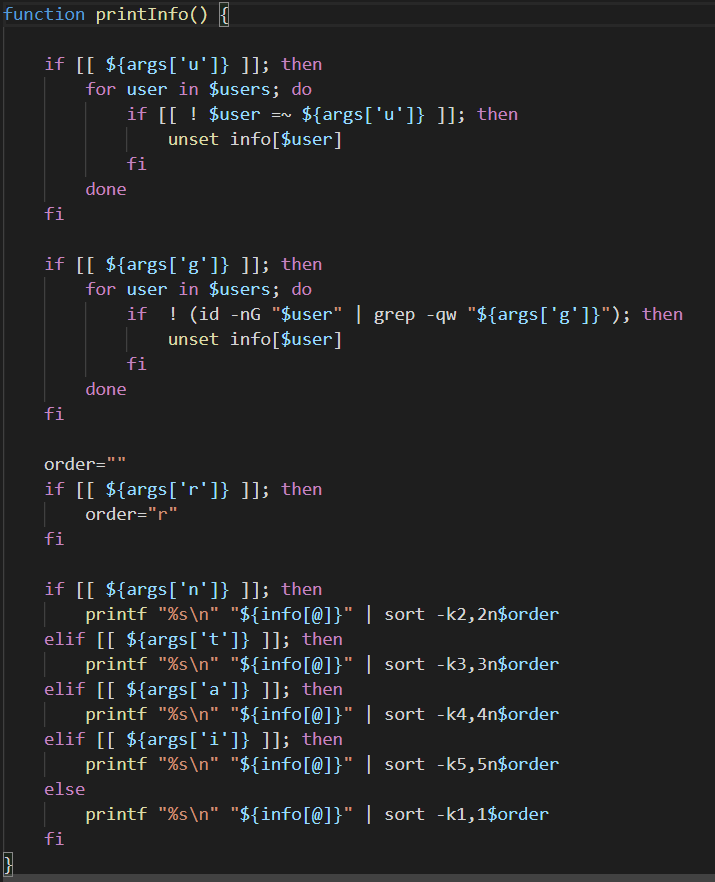
Percorrem-se os utilizadores do array **users[]** e utiliza-se de novo o comando last para obter a informação relevante, as opções -f, -s e -e são lidadas da mesma maneira que na função **getUsers().**

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**



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.

Fig. 8 – Função *getUsers()*

# **Metodologia usada para Solucionar o Problema**

**CompareStats.sh**

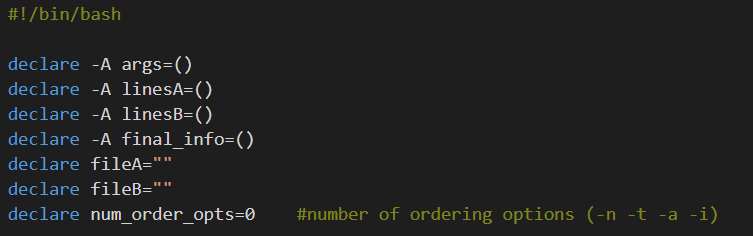


Fig. 9 – Fig. 1 – Declaração de arrays e variáveis.

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

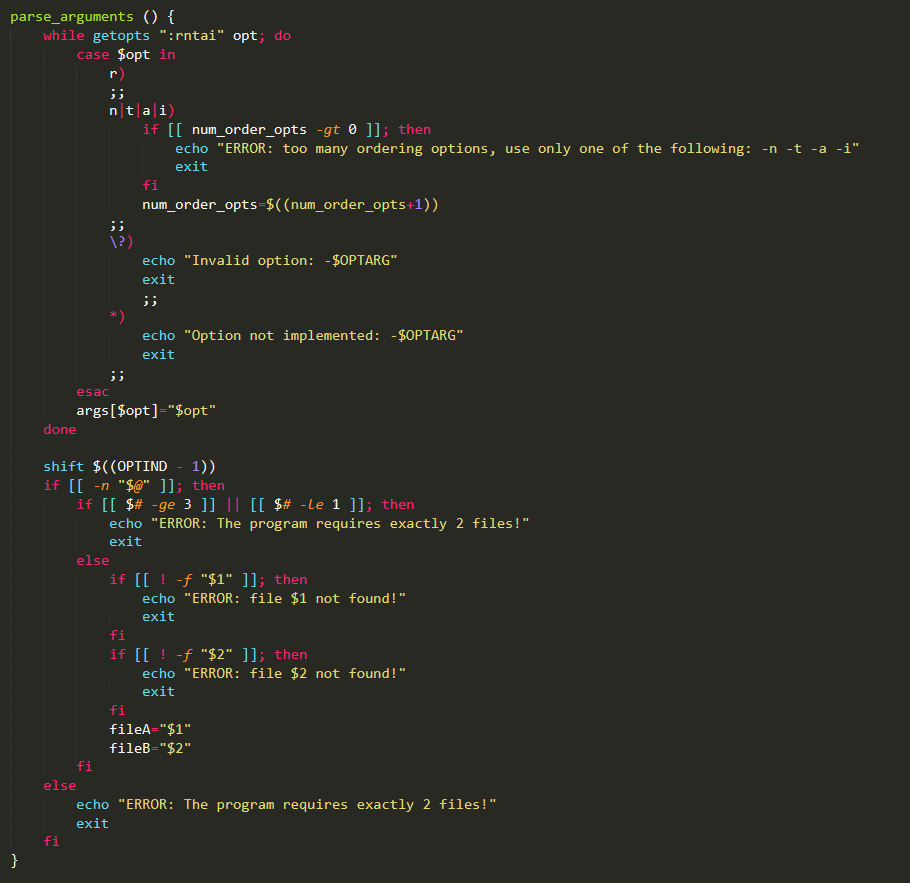


Fig. 10 – Função parse\_arguments()

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!")

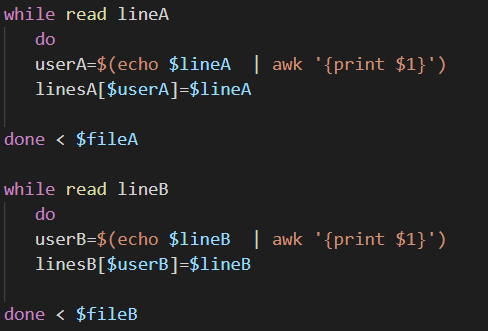


Fig. 11 – Leituras dos ficheiros

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.**

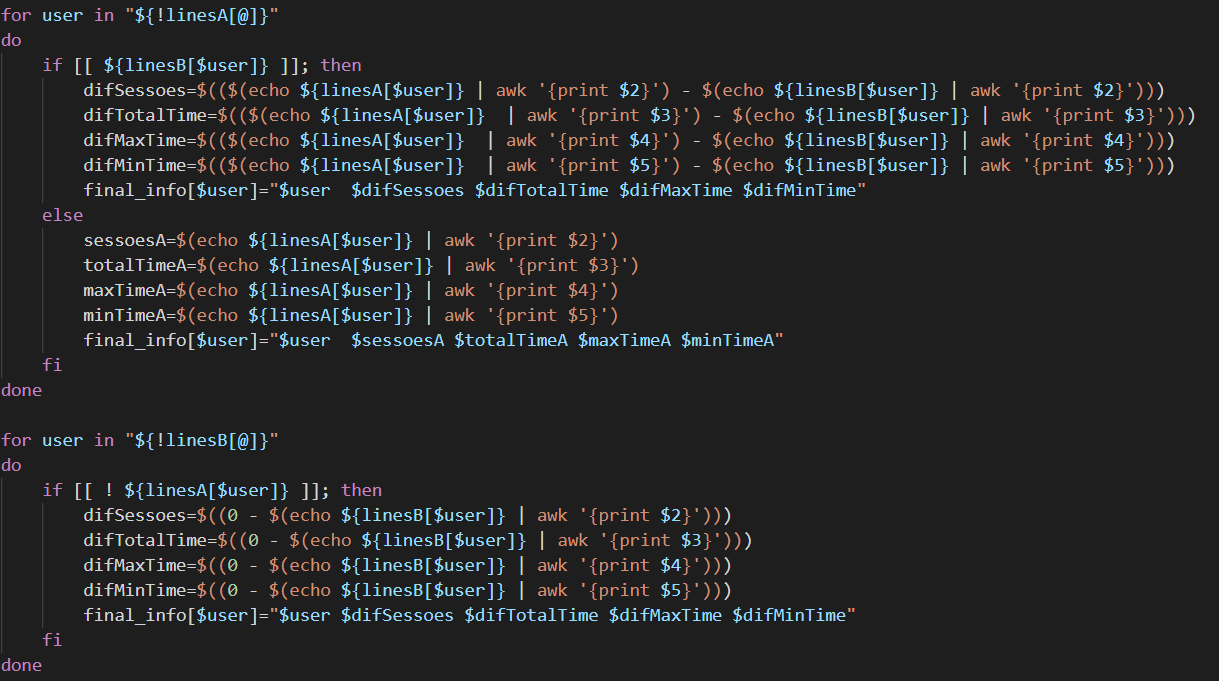


Fig. 12 – Operações entre ficheiros

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.**

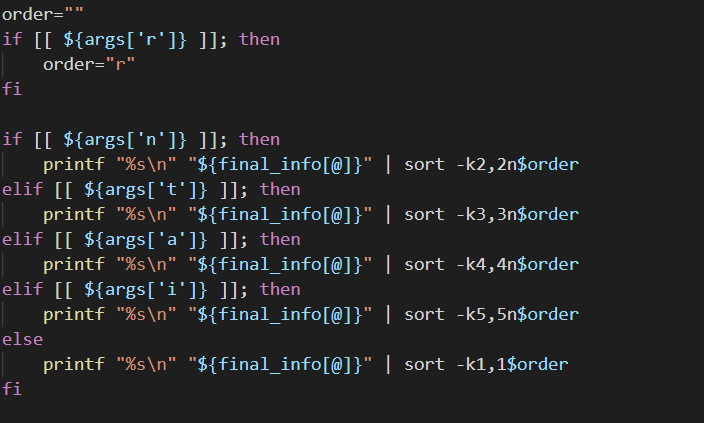


Fig. 13 – Verificação dos argumentos de ordenação

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**

# 

# 

# 

# 

Fig. 14 – ./userstats.sh

Fig. 15 – $./userstats.sh

Fig. 15 – ./userstats.sh -f wtmp.1

# 

# 

Fig. 16 – $./userstats.sh -u “sop.\*”

Fig. 17 –$./userstats.sh -g sop

# 

# 

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

# 

Fig. 20 – $./userstats.sh -n -u “sop.\*”

Fig. 19 – $./userstats.sh -t -u “sop.\*”

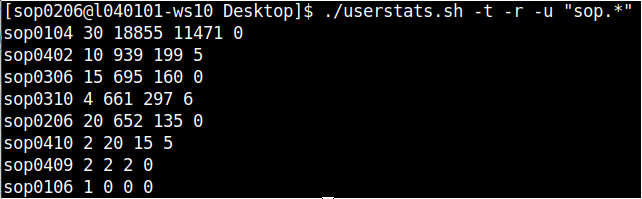
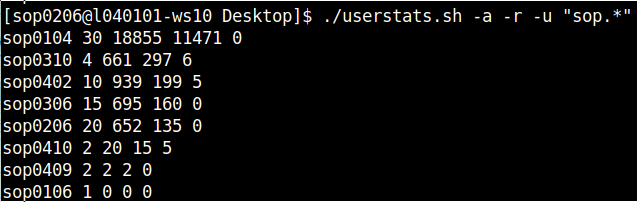


Fig. 22 – $./userstats.sh -a -r -u “sop.\*”

Fig. 21 – $./userstats.sh -t -r -u “sop.\*”

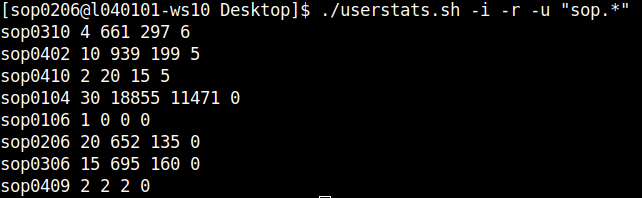
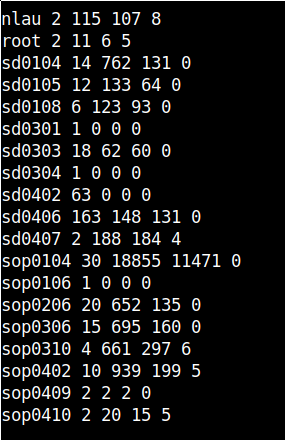


Fig. 23 – $./userstats.sh -i -r -u “sop.\*”





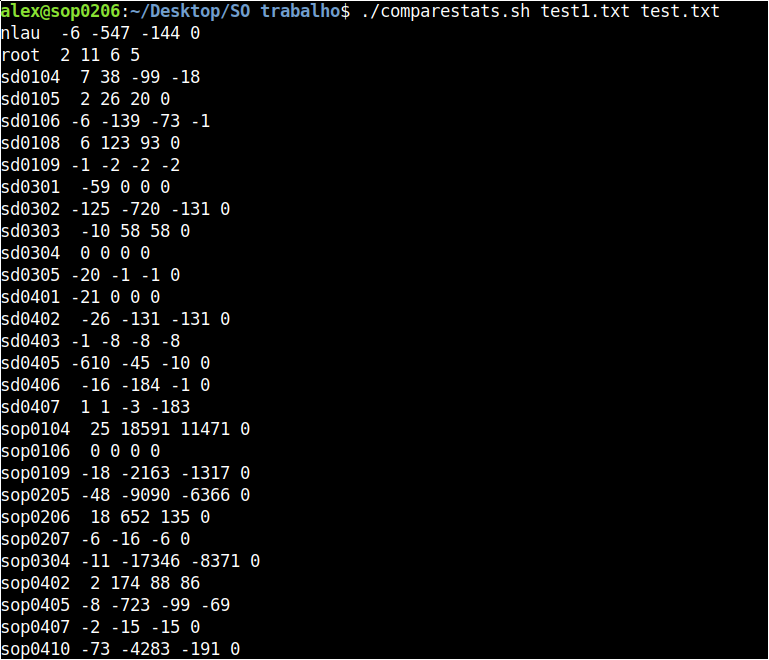


Fig. 24 – $./userstats.sh -n “.\*” > userstats\_123

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

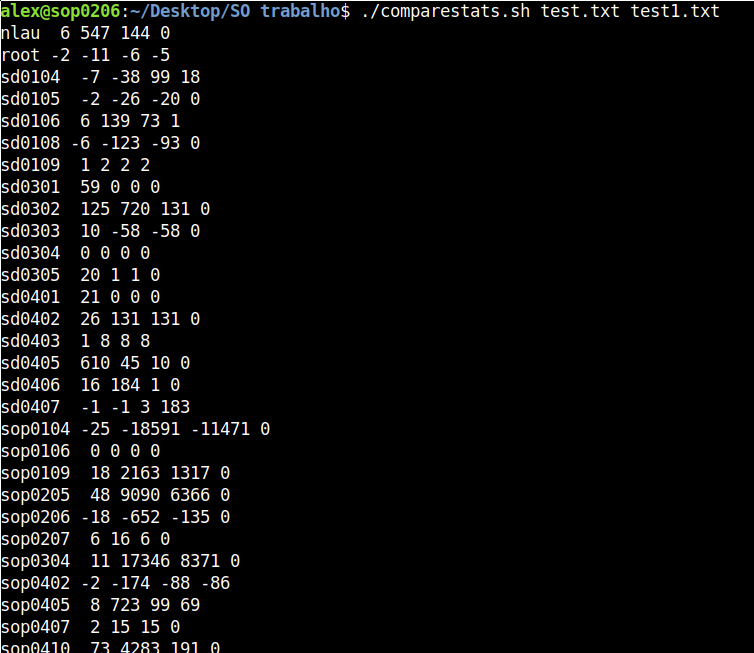
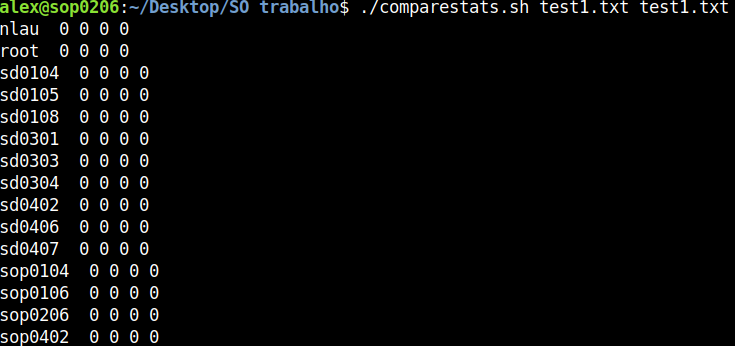


Fig. 27 – $./comparestats.sh test1.txt test1.txt

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

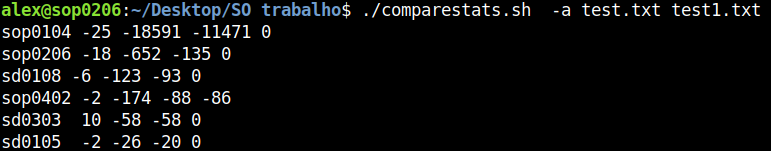


Fig. 28 – $./comparestats.sh -a test.txt test1.txt

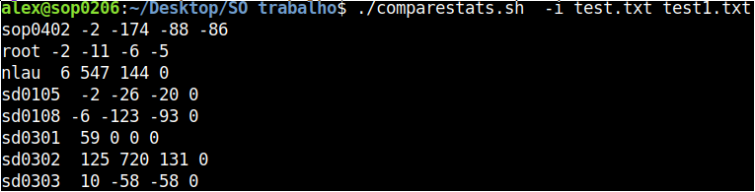


Fig. 29 – $./comparestats.sh -i test.txt test1.txt

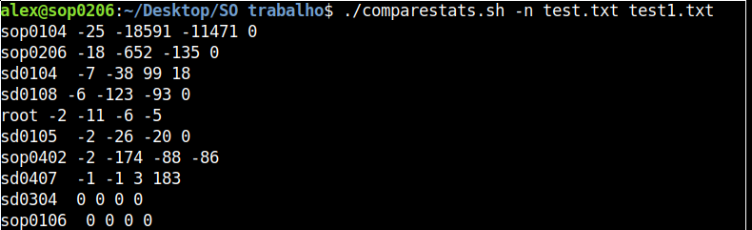


Fig. 30 – $./comparestats.sh -n test.txt test1.txt

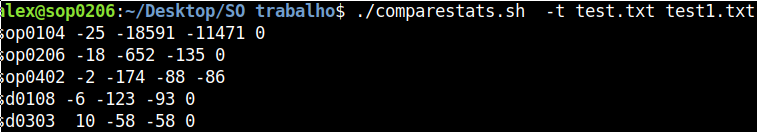


Fig. 31 – $./comparestats.sh -t test.txt test1.txt

# **Testes de Erros**



Fig. 32 – Too many ordering options



Fig. 33 ERRO: parameters are not valid: \_\_\_



Fig. 34 – -f file not found



Fig. 35 – -x invalid option



Fig. 36 – option needs argument

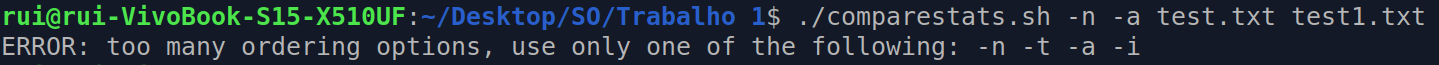


Fig. 37 – Too many ordering options

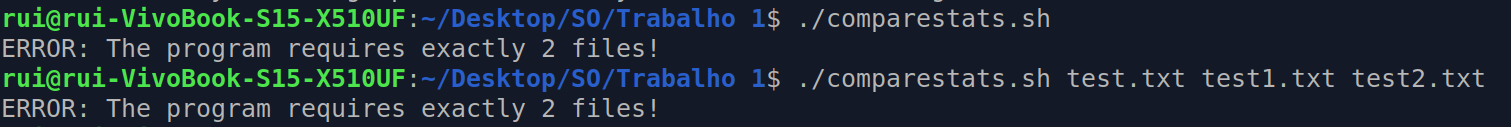


Fig. 38 – Wrong file amount

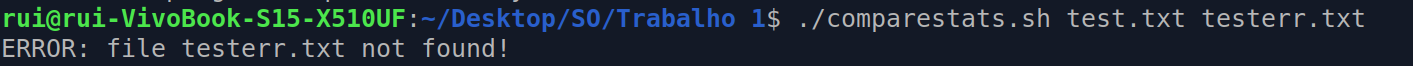


Fig. 39 – File not found

# 

# 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/>