universidade de aveiro
deti departamento de eletrónica, telecomunicações e informática

Trabalho 1

Gestão de armazenamento: Monitorização do espaço ocupado

Sistemas Operativos
Ano Letivo: 2023/2024
Prof. José Nuno Panelas Nunes Lau

Trabalho realizado por:
Rodrigo Abreu, Nº113626
João Neto, Nº113432

Índice

Introdução

O objetivo deste relatório é o desenvolvimento de scripts em bash que permitam o monitorizar o espaço ocupado em disco, e a sua variação ao longo do tempo, por ficheiros, o que desta forma, torna mais fácil a gestão do armazenamento.

O *script* ***spacecheck.sh*** permite a visualização do espaço ocupado pelos ficheiros selecionados nas diretorias que lhe são passadas como argumento e em todas as subdiretorias destas.

A seleção dos ficheiros a contabilizar, pode ser filtrada através das flags: **-n**, -s e **–d,** seguidas de um certo valor, cujas funcionalidades são visualizar, respetivamente, apenas os ficheiros com um certo nome, apenas os ficheiros até uma certa data de modificação limite e apenas os ficheiros com um tamanho maior que o passado como argumento.

Estes resultados, aparecem por omissão ordenados pelo tamanho de cada ficheiro de forma decrescente, mas podem ser representados de formas diferentes, nomeadamente ordenados por nome, através da flag **–a**, com a ordem invertida através da flag **–r** e com o número de resultados limitado através da flag **–l**.

O *script* ***spacerate.sh*** compara 2 ficheiros resultantes da execução do script anterior e, para todas as diretorias, representa a diferença entre os espaços ocupados. O resultado pode também ser ordenado de formas idênticas às do comando ***spacecheck.sh***.

Ao longo do relatório, vai ser demonstrado o nosso método de abordagem perante o desenvolvimento dos respetivos programas, bem como amostras do funcionamento dos mesmos.

O programa *spacecheck.sh*

Este programa foi dividido e resolvido por partes sendo desenvolvidas pela sua dependência umas pelas outras.

No início do programa é efetuada a declaração dos *arrays* e das variáveis globais que vão ser essenciais no decorrer do programa.

```
#!/bin/bash

                                        #DECLARAÇÃOO DE ARRAYS/DICIONÁRIOS
declare -A dict                         #dicionário que contém como chaves os diretórios e como values o espaço ocupado pelos ficheiros
declare -a allfiles                     #array que contém todos os ficheiros já analisados pela função dicarr
declare -a dire                         #array que contém todos os inputs do utilizador
declare -a prefix_before                #array que contém todos os prefixos iniciais (com repetição)
declare -a sort_prefix                  #array que contém todos os prefixos (sem repetição)

                                        #DECLARAÇÃO DE VARIÁVEIS
declare a=false                         #assume true se (-a) for um argumento
declare r=false                         #assume true se (-r) for um argumento
declare size=false                      #assume true se (-s) for um argumento
declare sufix=false                     #assume true se (-n) for um argumento
declare has_l=false                     #assume true se (-l) for um argumento
declare fd=false                        #assume true se (-d) for um argumento
declare files=false                     #assume true se (-s, -n ou -d) forem argumentos
declare inputs="$@"                     #todos os inputs do utilizador
declare f=""
declare l=0
declare s=0
declare n_linhas=0
declare n=""
declare date1=$(date "+%b %d %H:%M")
declare today=$(date +'%Y%m%d')
countargs=0
```

Fig 1.1: Declaração de variáveis

1ª parte: Ler e organizar num array associativo o diretório (A) e todos os seus subdiretórios com o espaço dos ficheiros que cada um contém respetivamente.

```
dicarr(){
        local current_directory="$1"
        local full_name="${current_directory}"
        if [ -d "$full_name" ]; then
                if [[ ! -n ${dict["$full_name"]} ]]; then
                        find "$full_name" -maxdepth 1 -type f -print0 2>/dev/null > discard.txt
                        if [ $? -eq 1 ]; then
                                dict["$full_name"]="NA"
                                return 1
                        else
                                dict[$full_name]=0
                        fi
                fi
```

Fig 1.2: Função dicarr_P1_Diretórios

```
while IFS= read -r -d '' subdir; do
        dicarr "$subdir"
done < <(find "$current_directory" -maxdepth 1 -mindepth 1 -print0)
```

Fig 1.3: Função dicarr_P1_Recursividade

A função ***dicaar()*** é uma função recursiva (*Fig 1.3*), usando um loop "while" e a saída do comando "find", que guarda a entrada da função em *current_directory* e em seguida calcula o espaço que este ocupa no disco (guardando também o seu nome completo).

Caso o *full_name* corresponda a um diretório e este ainda não estiver em *dict,* procede-se à verificação de permissões no diretório, caso não as tenhamos guardamo-lo como key no array associativo e associamos-lhe o valor *"NA"*, se as tivermos guardamo-lo como key no array associativo e associamos-lhe o valor 0.

```
else
        local disk_usage=$(du -b "$current_directory" | awk '{print $1}')
        if [[ ! " ${allfiles[*]} " =~ " $full_name " ]];then
                for key in "${!dict[@]}"; do
                        if [[ " $full_name " == *"$key"* ]]; then
                                if [ "$files" = true ]; then
                                        if [ "$sufix" = true ]; then
                                                if [ -f "$full_name" ] && [[ "$full_name" =~ ^$n$ ]]; then
                                                        ((dict[$key] += disk_usage))
                                                fi
                                        fi
                                        if [ "$fd" = true ]; then
                                                LC_TIME=en_US.UTF-8 last_modified=$(date -d "$(stat -c %y "$full_name")" "+%b %d %H:%M")
                                                LC_TIME=en_US.UTF-8 timestamp1=$(date -d "$last_modified" "+%s")
                                                LC_TIME=en_US.UTF-8 timestamp2=$(date -d "$date1" "+%s")
                                                if [ "$timestamp1" \< "$timestamp2" ]; then
                                                        ((dict[$key] += disk_usage))
                                                fi
                                        fi
                                        if [ "$size" = true ]; then
                                                if [ ! "$disk_usage" -lt "$s" ]; then
                                                        ((dict[$key] += disk_usage))
                                                fi
                                        fi
                                else
                                        ((dict[$key] += disk_usage))
                                fi
                        fi
                done
                allfiles+=("$full_name")
        fi
fi
```

Fig 1.4: Função dicarr_P2_Leitura de ficheiros

Caso o *full_name* não seja um diretório:

- No primeiro **if**, verifica se o *full_name* já passou como variável anteriormente. Vendo se este corresponde a algum elemento do array *allfiles* que guarda todos os ficheiros que já correu;
- No loop **for** e no segundo **if**, percorre o *dict* para encontrar todas as *key* que correspondem a caminhos válidos para o ficheiro, para posteriormente poder adicionar o espaço desse ficheiro;
- No terceiro **if**, verifica se "*files = true"* caso seja é sinal que uma ou mais das flags **-d**, **-s** ou **-n** foram usados pelo utilizador, estes que vão afetar o tamanho a contabilizar de cada diretório;
    - ➔ Caso "*sufix = true"* (**-n**) só vão ser contabilizados os ficheiros que terminam em *$n* (target do **-n**);
    - ➔ Caso "*fd = true"* (**-d**) só vão ser contabilizados os ficheiros cuja data de modificação não ultrapassa *$date1* (target do **-d**);
    - ➔ Caso "*size = true"* (**-s**) só vão ser contabilizados os ficheiros com espaço de disco ocupado superior ou igual a *$s* (target do **-s**);

2ª parte: Recolher as flags, que foram colocadas pelo utilizador e os seus respectivos targets.

```
args(){
        while getopts ":n:d:s:ral:" arg; do
                case $arg in
                        n)
                                n=$OPTARG
                                sufix=true
                                errordetector "$n"
                                files=true
                                countargs=$((countargs + 2))
                                ;;
                        d)
                                d=$OPTARG
                                fd=true
                                errordetector "$d"
                                LC_TIME=en_US.UTF-8 date1=$(date -d "$d" "+%b %d %H:%M")
                                files=true
                                countargs=$((countargs + 4))
                                ;;
                        s)
                                s=$OPTARG
                                size=true
                                errordetector "$s"
                                files=true
                                countargs=$((countargs + 2))
                                ;;
```

Fig 1.5: Função args_Flags que afetam a função dicarr

Nesta função o loop *while getopts* é usado para analisar as opções e argumentos da linha de comando.

Dependendo do *$arg* irão ser mudados/criados valores de diferentes variáveis:

- **-n** , o target do **-n** é guardado na variável $n, o valor do *sufix* é mudado para *true*, é feita a chamada à função **errordetector** (esta função serve para verificar se há algum erro com o target dado pelo utilizador) com a variável *$n*, de seguida muda o valor do *files* para *true* e adiciona ao *countargs* o número de elementos que leu (flag + target);
- **-d** , o target do **-d** é guardado na variável *$d*, o valor do *fd* é mudado para *true*, é feita a chamada à função **errordetector** com a variável *$d*, seguidamente muda o valor do *date1* (que originalmente é a data atual) para *$d* no formato correto (O *LC_TIME=en_US,UTF-8* foi usado pois o SO estava em português, o que causava problemas ao ler a data*)*, de seguida muda o valor do *files* para *true* e adiciona ao *countargs* o número de elementos que leu (flag + target)
- **-s** , o target do **-s** é guardado na variável *$s*, o valor do *size* é mudado para *true*, é feita a chamada à função **errordetector** com a variável *$s*, de seguida muda o valor do *files* para *true* e adiciona ao *countargs* o número de elementos que leu (flag + target);

```
r)
        r=true
        countargs=$((countargs + 1))
        ;;
a)
        a=true
        countargs=$((countargs + 1))
        ;;
l)
        l=$OPTARG
        has_l=true
        errordetector "$l"
        countargs=$((countargs + 2))
        ;;
\?)
        echo "Um dos argumentos não existe" >&2
        exit
        ;;
:)
        echo "Opção -$OPTARG requer um argumento." >&2
        exit
        ;;
esac
```

Fig 1.6: Função args_Flags que afetam a função printhere

- **-r** e **-a**, mudam as respectivas variáveis para *true* e adicionam ao *countargs* o número de elementos que leu (flag). Estas flags não necessitam de targets.
- **-l** , o target do **-l** é guardado na variável *$l*, o valor do *has_l* é mudado para *true*, é feita a chamada à função ***errordetector*** com a variável *$l* e adiciona ao *countargs* o número de elementos que leu (flag + target);

3ª parte: Apresentação dos resultados

```
printhere(){
        printf "Size Name $today $inputs\n"
        n_linhas=${#dict[@]}
        if [ "$has_l" = true ]; then
                if [ "$l" -le "$n_linhas" ]; then
                        n_linhas=$l
                fi
        fi
```

Fig 1.7: Função printhere - Número de linhas

Por omissão o número de linhas é igual à quantidade de elementos do *dict*, mas caso a variável *has_l* assumir o valor *true*, o número de linhas (caso este não seja maior que o valor default) é igual à variável *$l*.

```
if [ "$r" = true ]; then
        if [ "$a" = true ]; then
                for key in "${!dict[@]}"; do printf "%s %s\n" "${dict["$key"]}" "$key"; done | sort -k2,2r | while read -r line; do
                        if [[ $sum == $n_linhas ]];then
                                break
                        fi
                        space=$(echo "$line" | awk '{print $1}')
                        dir=$(echo "$line" | cut -d" " -f2-)
                        printF "$dir" "$space"
                        sum=$((sum + 1))
                done
        else
                for key in "${!dict[@]}"; do printf "%s %s\n" "${dict["$key"]}" "$key"; done | sort -k1,1n | while read -r line; do
                        if [[ $sum == $n_linhas ]];then
                                break
                        fi
                        space=$(echo "$line" | awk '{print $1}')
                        dir=$(echo "$line" | cut -d" " -f2-)
                        printF "$dir" "$space"
                        sum=$((sum + 1))
                done
        fi
```

Fig 1.8: Função printhere - Ordenação

Nesta parte da função se *$r* igual a *true* serão feitas as seguintes ordenações:

- Se *$a* igual a *true* a ordenação feita será o inverso da ordem alfabética, o **for** loop itera o *dict* por cada par valor-chave, considerando a chave pela ordem desejada, este loop vai acontecer até o número de linhas ser atingido, enquanto não for atingido *space* vai ficar com o valor e *dir* com a chave (cut -d" " -f2- , divide a linha por " " e -f2- escolhe o segundo elemento e os seus subsequentes), é chamada a função ***printF*** (onde se faz o print), e finalmente é adicionado 1 à variável *sum*;
- Caso *$a* seja *false* apenas muda o objeto de ordenação, aqui a ordenação será feita pela ordem crescente do valor.

```
else
        if [ "$a" = true ]; then
                for key in "${!dict[@]}"; do printf "%s %s\n" "${dict["$key"]}" "$key"; done | sort -k2,2 | while read -r line; do
                        if [[ $sum == $n_linhas ]];then
                                break
                        fi
                        space=$(echo "$line" | awk '{print $1}')
                        dir=$(echo "$line" | cut -d" " -f2-)
                        printF "$dir" "$space"
                        sum=$((sum + 1))
                done
        else
                for key in "${!dict[@]}"; do printf "%s %s\n" "${dict["$key"]}" "$key"; done | sort -k1,1nr | while read -r line; do
                        if [[ $sum == $n_linhas ]];then
                                break
                        fi
                        space=$(echo "$line" | awk '{print $1}')
                        dir=$(echo "$line" | cut -d" " -f2-)
                        printF "$dir" "$space"
                        sum=$((sum + 1))
                done
        fi
fi
```

Fig 1.9: Função printhere - Ordenação

Caso o *$r* assuma o valor *$false*:

- Se *$a* igual a *true* a ordenação feita será em ordem alfabética, o **for** loop itera o *dict* por cada par valor-chave, considerando a chave pela ordem desejada, este loop vai acontecer até o número de linhas ser atingido, enquanto não for atingido *space* vai ficar com o valor e *dir* com a chave (cut -d" " -f2- , divide a linha por " " e -f2- escolhe o segundo elemento e os seus subsequentes), é chamada a função ***printF*** (onde se faz o print), e finalmente é adicionado 1 à variável *sum*;
- Caso *$a* seja *false* apenas muda o objeto de ordenação, aqui a ordenação será feita pela ordem decrescente do valor.

```
printF(){
        local path="$1"
        local disk="$2"
        if [ "$disk" == "-1" ]; then
                disk="NA"
        fi
        for c in "${sort_prefix[@]}"; do
                if [[ "$path" == *"$c"* ]]; then
                        prefix=$c
                fi
        done
        path="${path#$prefix}"
        path="${path#/}"
        printf "%s %s\n" "$disk" "$path"
}
```

Fig 1.10: Função printF - Output

Esta função recebe as variáveis *path* e *disk* e faz print dos resultados:

- O **for** loop, itera pelo array *sort_prefix* e caso encontre um *$c* que esteja contido no *$path* assume-o como prefixo e retira-o do *$path*, dando seguidamente print do resultado

4ª parte: A função main

```
spacecheck(){
        args "$@"
        for f in "$@"; do
                dire+=($f)
                errordetector "diretório?"
                if [ -d "$f" ]; then
                        prefix_before+=("$(dirname "$(realpath -e "$f")")")
                        dicarr "$(realpath -e "$f")" "$s" "$n" "$fd"
                fi
        done
        for p in "${prefix_before[@]}"; do
                found=false
                for c in "${sort_prefix[@]}"; do
                        if [[ "$p" == *"$c"* ]]; then
                                found=true
                                break
                        fi
                done
                if [ "$found" == false ]; then
                        sort_prefix+=("$p")
                fi
        done
        printhere "$@"
}

spacecheck "$@"
```

Fig 1.11: Função spacechek (main)

A função ***spacechek*** é a função "main" do nosso código:

- Começa por fazer a chamada à função ***args*** dando lhe, como argumentos, todos os inputs escritos pelo utilizador;
- Seguidamente, através de um **for** loop, percorre todos esses inputs e adiciona-os no array *dire* e faz uma chamada à função ***errordetector*** (que irá verificar se o diretório é válido), caso o diretório seja válido guarda em *prefix_before* o prefixo do *realpath* do *$f* , em seguida faz a chamada à função ***dicarr*** com o *realpath* do *$f* e as variáveis *$s, $n* e $*fd* como argumentos;
- O **for** loop seguinte itera pelo array *prefix_before*, assumindo sempre *found=false* e caso encontre em *sort_prefix* um *$c* que contenha *$p*, *found* passa a assumir *true* e sai do respectivo loop, ao finalizar o segundo loop e *found* continuar a assumir *false* é adicionado ao array *sort_prefix* o valor *$p;*
- Finalmente é feita a chamada à função ***printhere***.

5ª parte: Detetar erros feitos nos argumentos escritos pelo utilizador

```
errordetector(){
        local check="$1"
        if [[ ! ( "$1" =~ ^[0-9]+$ ) &&  "$has_l" == true || ! ( "$s" =~ ^[0-9]+$ ) &&  "$size" == true ]]; then
                echo "Error: Tem a certeza que o target do -l ou -s só contem um ou mais dígitos"
                exit 1
        elif [[ -n "$n" && "$n" =~ [0-9] && "$sufix" == true ]]; then
                echo "Error: Tem a certeza que o target do -n é uma string"
                exit 1
        elif ! date -d "$d" &>/dev/null && [[ "$fd" == true ]]; then
                echo "Error: Tem a certeza que o target do -d é uma data válida (não te esqueças das aspas)"
                exit 1
        else
                for ((i = $countargs; i < ${#dire[@]}; i++)); do
                        if [ ! -d "${dire[i]}" ]; then
                                echo "Error: '${dire[i]}' ------ diretório inválido"
                                echo "Aviso: Tenha a certeza que os diretórios são os últimos argumentos que escreve"
                                exit 1
                        fi
                done
        fi

}
```

Fig 1.12: Função errordetector - Detetor de erros

Esta função serve para verificar se o utilizador cometeu algum erro nos inputs:

- No primeiro **if**, a função verifica se quando *size* ou *has_l* é igual a *true* a respectiva variável é apenas composta por dígitos;
- No primeiro **elfi**, a função verifica se *$n* existe e se existir verifica que esta é uma string, isto caso *$sufix=true*;
- No segundo **elfi**, caso *$fd=true* a função verifica se a data (*$d*) inserida pelo utilizador é válida;
- Caso nenhum dos outros aconteça, a função verifica se os elementos do array *dire*, após as flags e os seus respectivos targets, são diretórios válidos.

Resultados

```
joaoneto@joaoneto-OMEN-Laptop-15-en1xxx:~/Documents$ ./spacecheck.sh test_a1
Size Name 20231110 test_a1
66508 test_a1
40368 test_a1/aaa
9544 test_a1/rrr
7240 test_a1/aaa/zzzz
```

Fig 1.13: Exemplos_Default

```
joaoneto@joaoneto-OMEN-Laptop-15-en1xxx:~/Documents$ ./spacecheck.sh -a -r test_
a1
Size Name 20231110 -a -r test_a1
9544 test_a1/rrr
7240 test_a1/aaa/zzzz
40368 test_a1/aaa
66508 test_a1
```

Fig 1.14: Exemplos_Ordenação

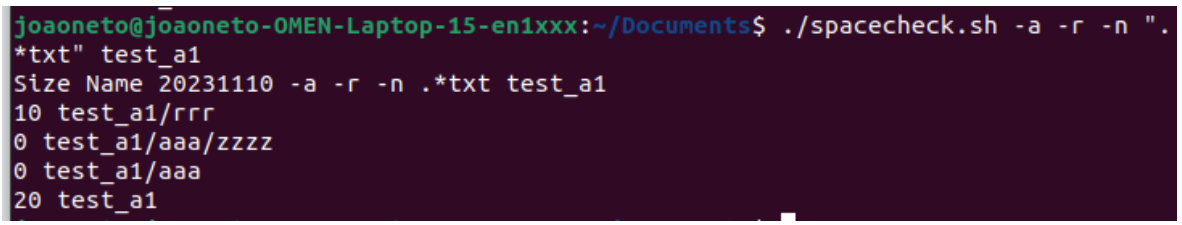

Fig 1.15: Exemplos_Ordenação&Name

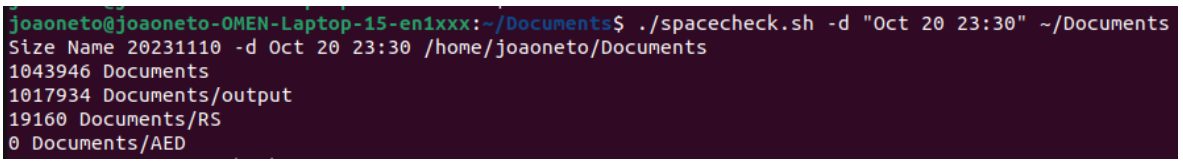

Fig 1.16: Exemplos_Date

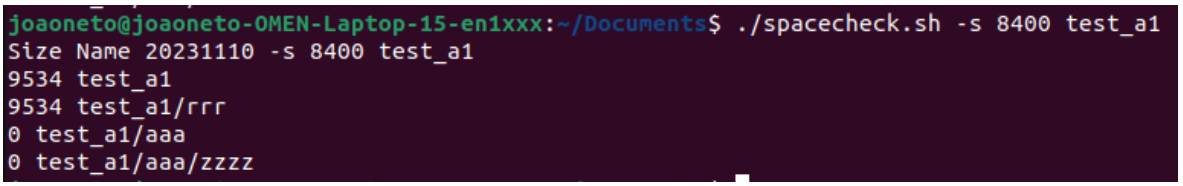

Fig 1.17: Exemplos_Size

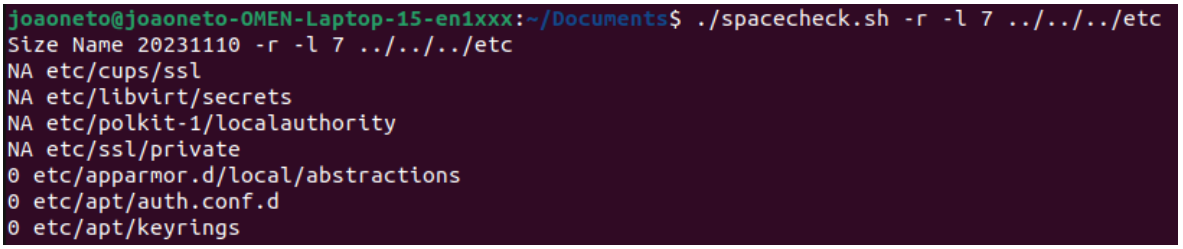

Fig 1.18: Exemplos_Ordenação&Limit&NA

O programa *spacerate.sh*

Para o desenvolvimento deste programa, decidimos dividi-lo por partes e encarar cada uma individualmente.

No início do programa é efetuada a declaração dos arrays e das variáveis globais que vão ser essenciais no decorrer do programa.

```
#Declaração de arrays
declare -A output1           #array que guarda o output de uma execução do script (spacecheck.sh)
declare -A output2           #array que guarda o output de uma execução do script (spacecheck.sh)
declare -A removed_output    #array que guarda os diretorios e atribui valor true aos removidos
declare -A new_output        #array que guarda os diretorios do segundo ficheiro e atribui o valor true aos novos ficheiros
declare -a files             #array que guarda os ficheiros passados como argumento
declare -A final_output      #array que guarda o output final

#Declaração de variáveis
declare r=false              #por default a ordem não é invertida
declare a=false              #por default os ficheiros não estão ordenados por nome
declare type_sort="-k1,1nr"  #tipo de ordenação, por omissão é ordenado por ordem decrescente de tamanho
```

Fig 2.1 - Declaração de arrays e variáveis globais.

1ª Parte: Ler os argumentos da função e especificar o papel de cada um deles.

Para isto, foi necessário fazer com que o programa conseguisse lidar com argumentos indesejáveis e erros do utilizador ao executar a função.

A função ***validate_files()*** garante que são apenas e só passados como argumentos 2 ficheiros, como se pode verificar através dos exemplos apresentados.

```
$ ./spacerate.sh ~/SO/Projeto1/SO_Project1/output1 ~/SO/Projeto1/SO_Project1/output2 ~/SO/Projeto1/SO_Project1/output3
```

Fig 2.2 - Execução do programa com 3 ficheiros passados como argumentos.

```
rodrigoabreu@pc-rodrigo-abreu:~/SO/Projeto1/SO_Project1
Não existem exatamente 2 argumentos do tipo ficheiro.
```

Fig 2.3 - Mensagem de erro resultante da execução da Fig 2.2.

A função args(), através do comando *getops*, caso detete as flags "-r" e "-a" e atribui o valor "true", às variáveis "r" e "a", isto que depois será útil para definir o tipo de ordenação dos resultados. Lida também com argumentos inválidos, alertando o utilizador.

```
args(){ #Esta função lida com os argumentos
        while getopts "ra" arg 2>/dev/null; do
                case $arg in
                        r)
                                r=true  #ativa a inversão da ordenação
                                ;;
                        a)
                                a=true #ativa a ordenação por ordem alfabetica
                                ;;
                        \?)
                                echo "Insira argumentos válidos. " >&2 #lida com argumentos inválidos
                                exit 1
                                ;;
                        esac
        done
}
```

Fig 2.3 - Função *args()*.

```
Insira argumentos válidos.
rodrigoabreu@pc-rodrigo-abreu:~/SO/Projeto1/SO_Project1$ ./spacerate.sh -b ~/SO/Projeto1/SO_Project1/output1 ~/SO/Projeto1/SO_Project1/output3
Insira argumentos válidos.
```

Fig 2.4 - Resultado da execução da função args() com argumentos inválidos.

Concluindo esta parte, foi necessário armazenar o conteúdo de cada ficheiro em estruturas de dados de forma a poderem ser analisados, e verificar a evolução que ocorreu de um ficheiro para o outro.

2ª Parte: Armazenamento do conteúdo dos ficheiros em estruturas de dados.

Através da função ***outputStorer()***, para ambos os ficheiros, é lido o conteúdo linha a linha, ignorando a primeira, e através do comando cut divide-se a linha em partes (tamanho e nome) que serão guardadas nos respetivos arrays associativos output1 e output2.

No ficheiro 1, são guardados os nomes dos diretórios com valor "true" no array associativo *removed_output.*

No ficheiro 2, são guardados os nomes dos diretórios com valor "true" no array *new_output,* caso nenhum diretório do ficheiro 2 tenha o mesmo nome que um diretório do ficheiro 1. Caso contrário, ao valor do respetivo diretório no array *removed_output* é atribuído o valor "false", pois se existe no ficheiro 2 um diretório com o mesmo nome, assume-se que ele não foi removido.

```
skip_line1=true

while IFS= read -r linha || [[ -n "$linha" ]]; do #leitura do ficheiro 1 linha a linha
        if [ "$skip_line1" = true ]; then #ignorar primeira linha
                skip_line1=false #as proximas linhas vão ser lidas
                continue

        else
                size=$(echo "$linha" | cut -d\  -f1) #linha dividida em 2, atribuindo a primeira parte à variável size
                name=$(echo "$linha" | cut -d\  -f2-) #linha dividida em 2, atribuindo a segunda parte à variável name
                output1[$name]=$size #guarda se num array associativo o nome do diretorio e o respetivo tamanho
                removed_output[$name]=true #guarda-se o nome do diretorio no array "removed_output" com variavel true
        fi
done < "${files[0]}"
```

Fig 2.5 - Loop da função outputStorer() que itera pelo ficheiro 1.

```
skip_line1=true

while IFS= read -r linha || [[ -n "$linha" ]]; do #leitura do ficheiro 1 linha a linha
        if [ "$skip_line1" = true ]; then #ignorar primeira linha
                skip_line1=false #as proximas linhas vão ser lidas
                continue

        else
                size=$(echo "$linha" | cut -d\  -f1) #linha dividida em 2, atribuindo a primeira parte à variável size
                name=$(echo "$linha" | cut -d\  -f2-) #linha dividida em 2, atribuindo a segunda parte à variável name
                output1[$name]=$size #guarda se num array associativo o nome do diretorio e o respetivo tamanho
                removed_output[$name]=true #guarda-se o nome do diretorio no array "removed_output" com variavel true
        fi
done < "${files[0]}"
```

Fig 2.6 - Loop da função outputStorer() que itera pelo ficheiro 2.

Concluído o armazenamento do conteúdo dos ficheiros, procede-se à análise.

3ª Parte: Análise da evolução do armazenamento dos diretórios.

A função ***outputComparer()*** vai proceder ao cálculo da variação do espaço ocupado em cada diretório e guardar essa informação no array associativo *final_output*.

Os arrays removed_output e new_output serão importantes neste processo. O primeiro, porque o cálculo do da diferença de armazenamento para os ficheiros removidos é efetuada de forma diferente. O segundo, é necessário para adicionar os novos ficheiros ao *final_output*.

```
outputComparer(){ #vai comparar o conteudo dos 2 ficheiros e adicionar de forma correta os valores ao array final_output
        for name1 in "${!output1[@]}"; do
                sizeout1=${output1[$name1]}
                removed=${removed_output[$name1]}
                if [ "$removed" == true ]; then
                        difference=$(( 0 - $sizeout1 )) #subtrai-se o tamanho total do diretorio caso este tenha sido removido
                        final_output[$name1]=$difference
                else
                        for name2 in "${!output2[@]}"; do
                                sizeout2=${output2[$name2]}
                                if [ "$name2" == "$name1" ]; then
                                        difference=$(( $sizeout2 - $sizeout1 ))  #calcula da variação do tamanho de cada diretorio
                                        final_output[$name2]=$difference
                                        new_output[$name2]=false #como os nome diretorios são iguais, $name2 não é diretorio novo (false)
                                fi
                        done
                fi
        done
        for name2 in "${!new_output[@]}"; do
                if [ ${new_output[$name2]} == true ]; then
                        final_output[$name2]=${output2[$name2]} #adicionar os diretorios novos  e o respetivo tamanho ao final_output
                fi
        done
}
```

FIg 2.7 - Função outputComparer()

4º Parte: Ordenação e apresentação dos resultados.

Quando a análise e o armazenamento dos dados são completados de forma acertada, procede-se à ordenação e apresentação dos resultados.

Por omissão os resultados são apresentados por ordem decrescente de tamanho, mas isto pode ser alterado na função ***typesort()***, de acordo com o valor das variáveis "r" e "a" como se pode ver na figura.

```
typesort(){ #esta função define o tipo de ordenção tendo em conta o valor das variaveis definido na função args
        if [ "$r" == true]; then
                if [ "$a" == true ]; then
                        type_sort="-k2,2r" #filtro por nome com a ordem invertida
                else
                        type_sort="-k1,1n" #filtro por tamanho por ordem crescente
                fi
        else
                if [ "$a" == true ]; then
                        type_sort="-k2,2" #filtro por nome com a ordem invertida
                fi
        fi
}
```

Fig 2.8 - Função typesort().

A ordenação e apresentação dos resultados são efetuados pela função ***printer()***. A ordenação depende da variável *type_sort* e pode ser das seguintes formas:

- Por ordem decrescente de tamanho dos diretórios.
- Por ordem crescente de tamanho dos diretórios.
- Por ordem alfabética.
- Por ordem alfabética invertida.

Para cada diretório do array *final_output*, se esse diretório tiver o valor "true" no array removed_output ou no array new_output será apresentado, respetivamente, "REMOVED" ou "NEW" à frente do seu resultado.

A função ***spacerate()*** passa os argumentos necessários para cada função e executa-as.

```
spacerate(){ #main, passa apenas os argumentos necessárioa cada função e executa-as
        validate_files "$@"
        args "$@"
        outputStorer
        outputComparer
        typesort
        printer
}

spacerate "$@" #execução do programa
```

Fig 2.9 - Função spacerate().

Seguem-se resultados da execução do programa.

Resultados

Foram guardados os resultados de 2 execuções do programa ***spacecheck.sh***, em ficheiros de texto com os respectivos nomes, “output1” e “output2”, de modo a poder ser analisada a evolução do armazenamento dos diretorios no programa ***spacerate.sh*** .

Nas figuras, são apresentados os resultados da execução do programa para diferentes tipos de argumentos.

output1

```
1   Size Name 20231031 -a /home/rodrigoabreu/Documentos
2   40 Documentos
3   4 Documentos/Airetorio
4   16 Documentos/Ciretorio
5   8 Documentos/Diretorio
6   12 Documentos/Ziretorio
7
```

Fig 2.10 - Output1

output3

```
1   Size Name 20231031 -a /home/rodrigoabreu/Documentos
2   32 Documentos
3   8 Documentos/Ciretorio
4   8 Documentos/Diretorio
5   4 Documentos/Riretorio
6   12 Documentos/Ziretorio
7
```

Fig 2.11 - Output2

```
rodrigoabreu@pc-rodrigo-abreu:~/SO
SIZE NAME
4 Documentos/Riretorio NEW
0 Documentos/Diretorio
0 Documentos/Ziretorio
-4 Documentos/Airetorio REMOVED
-8 Documentos
-8 Documentos/Ciretorio
rodrigoabreu@pc-rodrigo-abreu:~/SO
```

Fig 2.11 - Execução da função spacerate.sh com os ficheiros “Output1” e “Output2” como argumentos sem filtros de ordenação.

```
rodrigoabreu@pc-rodrigo-abreu:~/SO
SIZE NAME
-8 Documentos
-8 Documentos/Ciretorio
-4 Documentos/Airetorio REMOVED
0 Documentos/Diretorio
0 Documentos/Ziretorio
4 Documentos/Riretorio NEW
rodrigoabreu@pc-rodrigo-abreu:~/SO
```

Fig 2.12 - Execução da função spacerate.sh com os ficheiros “Output1” e “Output2” como argumentos e o filtro de ordenação “-r”.

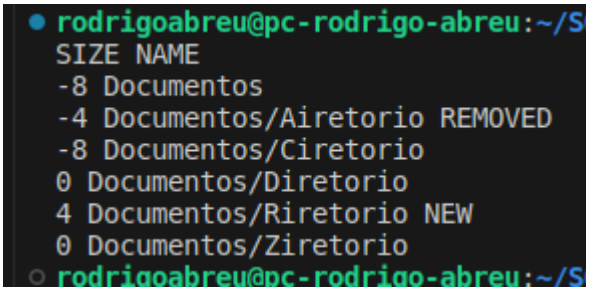

Fig 2.13 - Execução da função spacerate.sh com os ficheiros "Output1" e "Output2" como argumentos e o filtro de ordenação "-a".

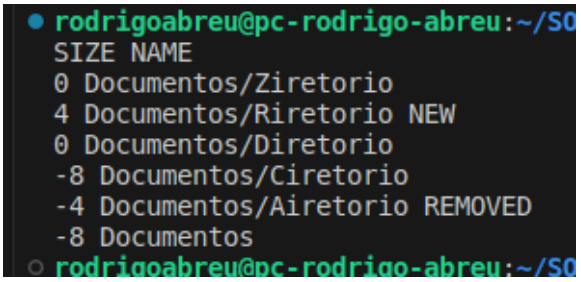

Fig 2.12 - Execução da função spacerate.sh com os ficheiros "Output1" e "Output2" como argumentos e os filtros de ordenação "-r" e "-a".

Conclusão

Este projeto mostrou-se desafiante, principalmente no início, no desenvolvimento dos programas, mas à medida que íamos resolvendo os problemas, ficámos mais ágeis com a linguagem *bash*. Além disto, foi importante para obtermos um conhecimento mais aprofundado, sobre a forma como o computador lida com armazenamento dos diretórios, dos subdiretórios e dos ficheiros,, isto que será de grande importância no nosso futuro na área da informática.

Bibliografia

- Guiões práticos da disciplina.
- https://stackoverflow.com/questions/16483119/an-example-of-how-to-use-getopts-in-bash
- https://guialinux.uniriotec.br/du/
- https://www.geeksforgeeks.org/cut-command-linux-examples/
- https://www.cyberciti.biz/faq/bash-scripting-using-awk/
- https://www.geeksforgeeks.org/sort-command-linuxunix-examples/
- https://www.geeksforgeeks.org/stat-command-in-linux-with-examples/
- https://www.geeksforgeeks.org/bc-command-linux-examples/
- https://unix.stackexchange.com/questions/9496/looping-through-files-with-spaces-in-the-names
- https://askubuntu.com/questions/264283/switch-command-output-language-from-native-language-to-english
- https://www.hostinger.com/tutorials/grep-command-in-linux-useful-examples/
- https://phoenixnap.com/kb/linux-head
- https://www.geeksforgeeks.org/find-command-in-linux-with-examples/