



deti

universidade de aveiro
departamento de electrónica,
telecomunicações e informática

Gestão de armazenamento: Monitorização do espaço ocupado

Trabalho realizado por:

Eduardo Lopes Nº 103070

Tomás Brás Nº 112665

Sistemas Operativos

Prof. José Nuno Panelas Nunes Lau

Prof. Guilherme Campos

Ano Letivo 2023/2024

Índice

INTRODUÇÃO	3
MÉTODOS UTILIZADOS NA REALIZAÇÃO DO PROJETO.....	4
Divisão do projeto	4
Spacecheck.sh	5
1) Argumentos de entrada.....	5
2) Variáveis inicializadas	7
3) Cálculo do espaço ocupado	8
4) Print e formatação do output	10
5) Main e verificações	11
Spacerate.sh.....	12
1) Argumentos de entrada.....	12
2) Variáveis inicializadas	13
3) Cálculo do espaço ocupado	14
4) Print e formatação do output	17
VALIDAÇÃO DOS RESULTADOS OBTIDOS (ANÁLISE).....	18
1) Ficheiros teste	18
2) Resultados do spacecheck.....	19
3) Resultados do spacerate.....	20
4) Erros e Avisos	21
CONCLUSÃO	22
BIBLIOGRAFIA.....	23

INTRODUÇÃO

Este projeto teve como objetivo a criação de dois scripts em bash, “spacecheck.sh” e “spacerate.sh”. Estas duas funções permitem visualizar e controlar o espaço ocupado no disco por ficheiros e diretórios ao longo do tempo.

Com esta implementação, é facilitada a visualização do espaço ocupado no disco do computador, em bytes, por ficheiros das respectivas diretorias, que são passadas como argumento no primeiro script (“spacecheck.sh”). O segundo script (“spacerate.sh”) permite-nos visualizar a evolução do espaço ocupado pelas diretorias presentes nos dois ficheiros passados como argumento.

Relativamente ao primeiro script, a escolha dos ficheiros a contabilizar para o cálculo do espaço ocupado, pode ser feita de diversas maneiras. Podemos escolher os ficheiros através de uma expressão contida no nome do ficheiro (**option -n**), a partir de uma data máxima de modificação dos ficheiros (**option -d**), ou através do tamanho mínimo do ficheiro (**option -s**). A visualização do output deste script também pode ser ordenada pelo nome dos ficheiros (**option -a**) e por ordem inversa (**option -r**). Outra opção é limitar o número de linhas da tabela apresentadas (**option -l**).

O segundo script permite a ordenação do output por nome (**option -a**) e por ordem inversa (**option -r**).

Ao longo do relatório, são explicados os métodos utilizados em cada um dos scripts, ou seja, a forma de calcular o espaço ocupado no disco, a validação usada para os argumentos dados na entrada e a formatação dos dados de saída.



MÉTODOS UTILIZADOS NA REALIZAÇÃO DO PROJETO

Durante a elaboração deste projeto, utilizámos vários métodos que nos permitiram alcançar resultados válidos, tais como pretendidos, e resolver todos os erros de execução, que foram surgindo ao longo do tempo.

Divisão do projeto

Este projeto foi dividido em duas partes principais. Cada parte corresponde à criação de um script, o “spacecheck.sh” e o “spacerate.sh”. A implementação de cada script está subdividida em várias fases.

Em ambos os scripts foi necessário a declaração de variáveis globais, fundamentais para a validação e armazenamento dos argumentos de entrada, passados pelo utilizador; declaração de um array global é responsável pelo armazenamento do espaço ocupado pelos ficheiros e a criação de funções (directories(), files(), input(), space(), print() e main()) essenciais para a correta execução do código.

A maior parte dos métodos utilizados são comuns a ambos os scripts, pelo que começaremos por explicar o primeiro script. De seguida, explicaremos os métodos do segundo script que são diferentes do primeiro. A explicação dos métodos será feita pela ordem que o script executa os mesmos.

Spacecheck.sh

1) Argumentos de entrada

Ao executar o script “spacecheck.sh”, temos de inserir diferentes parâmetros de entrada que depois de validados terão diferentes ações e nos permitem organizar os dados para proceder ao cálculo do espaço ocupado.

Para este efeito desenvolvemos duas funções de validação. A função **directories()** que itera sobre os argumentos passados, verificando quais deles são diretórios. Isto é feito através de uma condição if dentro de um ciclo for. Se o argumento verificar a condição (ou seja, se for um diretório), a função directories chama a função space com esse diretório como argumento.

```
function directories() {  
    for i in "$@"; do  
        if [[ -d "$i" ]]; then  
            space "$i"  
        fi  
    done  
}
```

fig. 1 | Função para verificar se o argumento de entrada é um diretório

A outra função é **inputs()** que lê os argumentos introduzidos que podem alterar a saída do script. Nesta função utilizamos um ciclo while (while getopts) para processar os parâmetros colocados pelo utilizador no terminal. Escolhemos este ciclo porque permitiu-nos escolher os parâmetros de entrada válidos (estes encontram-se entre aspas). Os parâmetros que são sucedidos de dois pontos (:) necessitam de ter um segundo argumento a seguir ao mesmo. A variável opt representa o parâmetro que está a ser analisada pelo getopts em cada iteração.

Os argumentos válidos presentes no case são os seguintes:

- **-n** - Método que é precedida de uma expressão/string (de um tipo específico de ficheiro), fazendo com que seja listada informação apenas sobre o espaço desse tipo de ficheiros e respectivos caminhos nesse diretório. Quando não existe uma expressão devem ser contabilizados todos os ficheiros existentes nesse diretório. O argumento é guardado na variável **expresao**.
- **-r** - Método que não necessita de argumento. Aqui o argumento é guardado na variável **reverse**, que pode tomar o valor 0 ou 1 (0 para a listar as informações de forma default e 1 para reverter a ordem).

- **-a** - Método que não necessita de argumento. O argumento é guardado na variável **sort_name**, que pode tomar o valor 0 ou 1 (0 para a listar as informações de forma default e 1 para ordenar por nome dos ficheiros e diretorias).
- **-l** - Método que limita o número de linhas exibidas. Este método é precedido por um valor inteiro que corresponde ao número de linhas máxima que a tabela de output deve ter. O argumento é guardado na variável **limite**.
- **-d** - Método que filtra todos os ficheiros que fazem parte do diretório até uma data máxima de modificação escolhida pelo o utilizador. Os ficheiros com data de criação/modificação superior a esta, não serão exibidos. O argumento é guardado na variável **input_date**.
- **-s** - Método que é precedido de um tamanho mínimo do ficheiro. Os ficheiros com tamanho inferior não serão contabilizados no cálculo do espaço ocupado pelo respetivo diretório a que pertencem. O argumento é armazenado na variável **minimo**.

No final desta função introduziu-se o comando “shift \$((OPTIND - 1))” para garantir que no final de cada iteração do ciclo while, os parâmetros que já foram processados pelos getopt são excluídos.

```
function input() {
    while getopts "n:rad:s:l:" flag; do
        case $flag in
            n)
                #verifica se a string tem comprimento maior que zero
                if [ -n "$OPTARG" ]; then
                    expressao=$OPTARG
                fi
                ;;
            r)
                reverse=1
                ;;
            a)
                sort_name=1
                ;;
            d)
                #verifica se a data é válida
                input_date=$(date -d "$OPTARG" +%s 2>/dev/null)
                if [[ -z $input_date ]]; then
                    echo "Data inválida"
                    exit 1
                fi
                ;;
            s)
                #verificação se o número é inteiro positivo
                if [[ $OPTARG =~ $validation ]] && [[ $OPTARG -gt 0 ]]; then
                    minimo=$OPTARG
                else
                    echo "Número inserido inválido"
                    exit 1
                fi
            ;;
        esac
        shift $((OPTIND - 1))
    done
}
```

fig. 2 | Função input que pode de alterar a saída do script consoante a flag escolhida

```

1)
#verificação se o número é positivo
if [[ $OPTARG -gt 0 ]]; then
    limite=$OPTARG
else
    echo "Número inserido inválido"
    exit 1
fi
;;
*)
#caso seja passado um argumento não válido
echo "Opção inválida! A sair..."
exit 1
;;
esac
done
shift $((OPTIND - 1))
}

```

fig. 2.1 | Continuação da função input

2) Variáveis inicializadas

Foram inicializadas variáveis que servem para nos ajudar a garantir o bom funcionamento do programa. Todas as variáveis têm um parâmetro default atribuído, caso estas não fossem selecionadas pelo utilizador.

As variáveis `total_space`, `minimo`, `reverse` e `sort_name` encontram-se inicializadas por defeito com o valor “0”. Inicializar a variável `minimo` com “0” permite-nos que todos os ficheiros com tamanho superior a zero sejam contabilizados no cálculo do espaço ocupado. A variável `reverse` e `sort_name`, quando inicializadas com “0”, permitem que o output não tenha nenhum tipo de ordenação para além da que existe por defeito.

A variável `limite` foi inicializada com o valor “10000”. Isto permite que a tabela apareça no output na íntegra caso o utilizador não selecione esta opção.

A variável `expressao` foi inicializada com “*” para que todos os ficheiros sejam contabilizados no cálculo do espaço ocupado, caso o utilizador não passe como argumento uma expressão reguladora que identifica o tipo de ficheiros pretendidos.

A variável `validation` encontra-se inicializada com a expressão “^[0-9]+\$”. Esta variável permite-nos verificar se o valor introduzido na opção `s` é um inteiro. Ao longo da execução do programa, esta variável nunca é alterada.

A variável `input_date` é inicializada com a data e hora em segundos, do momento de execução do programa. Caso o utilizador não selecione esta opção, todos os ficheiros são contabilizados, pois a última data de criação/modificação destes é inferior à data por defeito.

A variável `IFS` é inicializada com a expressão “\n”. Esta variável é utilizada em ciclos `while` com a intenção de contabilizar a leitura de diretórios e ficheiros com espaço no nome.

```
declare total_space=0
declare minimo=0
declare limite=10000
declare expressao="*"
declare reverse=0
declare sort_name=0
declare validation='^[0-9]+$'
declare input_date=$(date +%s)
```

fig. 3| Declaração de variáveis para garantir o bom funcionamento do programa

3) Cálculo do espaço ocupado

Nesta fase de implementação, desenvolvemos a função **space()**. Como referido anteriormente, esta função é invocada pela função **directories** juntamente com um diretório passado como argumento.

A função **space** é responsável pelo cálculo do espaço ocupado pelos ficheiros nos seus respectivos diretórios. Para este efeito foi necessário declarar um array associativo (**space_array**) para armazenar os resultados. Este array permite-nos utilizar como index os diretórios aos quais o espaço ocupado calculado corresponde.

```
#declaração de arrays
declare -A space_array
```

fig.4 | Declaração de array associativo

Nesta função, começamos primeiro por identificar todos os subdiretórios do diretório passado como argumento. Para isso, utilizamos o comando **find** com a flag “-type d” (percorre todos os ficheiros dentro do diretório e só devolve aqueles que são diretórios) e armazenamos o resultado na variável **dires**.

De seguida, é executado um ciclo **while** com a variável **IFS**, que permite a leitura de diretórios com um espaço no nome. Neste ciclo começamos por inicializar a variável **total_space** com “0” com o efeito de descartar o valor atribuído a esta variável na iteração anterior.

Posteriormente é executado outro ciclo **while** tal como o primeiro, mas desta vez com a intenção de ler ficheiros com espaço no nome. Nesta etapa, existe a filtragem dos ficheiros contabilizados pelas opções escolhidas pelo utilizador. Relembramos que é possível filtrar por nome/tipo de ficheiro e data máxima de criação/modificação (nesta fase).

No próximo passo, o programa verifica os ficheiros dentro de uma diretoria aos quais não conseguimos determinar o tamanho. Para este objetivo utilizamos o comando **find** com a flag “-type f” e outras que permitem a filtragem acima descrita, associado com “>/dev/null 2>&1”, que por sua vez permite

descartar o output do comando find tal como possíveis mensagens de erro. A variável result é utilizada para guardar o status de saída do último comando executado, ou seja, do find.

Seguidamente, é feita uma comparação através de um if, onde se verifica se o valor atribuído à variável result é diferente de "0". Se a condição for verdadeira, verifica-se que não foi possível determinar o tamanho do ficheiro. Sendo assim, é guardado no array o valor "NA" associado ao respetivo diretório e o programa passa para a próxima iteração.

Caso o valor atribuído à variável result seja "0" (o que indica que o comando find foi executado com sucesso), o programa salta para o if seguinte. Este if começa por verificar se a variável não é um diretório. Se isso se verificar, o if é executado e o espaço usado, em bytes, por este ficheiro é atribuído à variável space. Isto é feito através do comando du associado com a flag -b (responsável por apresentar o espaço em bytes) e o comando awk que nos permite imprimir só a primeira coluna do resultado do comando du.

A seguir, é executado o comando if, que verifica se o espaço ocupado pelo ficheiro é superior ao valor introduzido pelo utilizador e guardado na variável minimo (relembrar que por defeito é "0"). Se esta condição se verificar, a variável total_space é incrementada com o valor do espaço ocupado pelo ficheiro através do uso do comando bc.

No final de cada iteração do segundo ciclo while, o valor atribuído à variável total_space é armazenado no array (space_array) associado ao diretório a que os ficheiros contabilizados pertencem. Com isto, conclui-se esta interação do primeiro ciclo while e o programa avança para a próxima e contínua este processo até iterar sobre todos os diretórios e subdiretórios.

```
#calcula o espaço ocupado pelos ficheiros de um diretório
function space() {
    dires=$(find "$1" -type d)

    while IFS= read -r -d '' dir; do
        total_space=0
        while IFS= read -r -d '' file; do

            #verifica se o directorio impossível de aceder ou determinar o tamanho dos seus ficheiros
            find "$dir" -type f -name "$expressao" ! -newermt "@$input_date" -print0 >/dev/null 2>&1
            result=$?
            if [[ $result -ne 0 ]]; then
                space_array["$dir"]="NA"
                continue
            fi

            if [[ ! -d "$file" ]]; then
                space=$(du "$file" | awk '{print $1}' | grep -oE '[0-9.]+')
                if [[ $space -ge $minimo ]]; then
                    total_space=$(echo "$total_space + $space" | bc)
                fi
            fi
        done < <(find "$dir" -type f -name "$expressao" ! -newermt "@$input_date" -print0)

        # Guardar os valores num array
        space_array["$dir"]=$total_space
    done < <(find "$1" -type d -print0)
}
```

fig. 5| Função space, responsável pelo cálculo do espaço que cada diretório ocupa

4) Print e formatação do output

Para esta etapa, foi desenvolvida a função **print()**. Esta função é invocada diretamente pela main juntamente com todos os argumentos introduzidos pelo utilizador. Esta função é responsável pela criação do output final (tabela).

A função começa por verificar se existe um diretório nos argumentos passados pelo utilizador. Isto é possível através da verificação do número de elementos presentes no array (space array). Se o array tiver zero elementos, ficamos a saber que a função space não foi executada, o que por sua vez indica que a condição dentro da função directories não foi verificada com nenhum argumento passado pelo utilizador, ou seja, nenhuma diretoria foi passada como argumento. Nesse caso o programa devolve a mensagem “Precisa passar um diretório como argumento” e termina a execução do programa.

Caso existam argumentos que são diretórios, a função prossegue para imprimir a tabela com os resultados, começando pelo cabeçalho onde estão todos os argumentos passados pelo utilizador.

Como referido anteriormente, a tabela pode ser ordenada por ordem alfabética (opção a) ou por ordem inversa de tamanho (opção r). Também é possível limitar o número de linhas imprimidas com a opção l seguida do número de linhas pretendidas. Para este efeito foram utilizadas duas condições if, sendo que a primeira testa se o valor atribuído à variável reverse é “1”. Se for a tabela é imprimida pela ordem inversa de tamanho. A outra condição verifica se o valor da variável sort_name é “1”. Se verificar, a tabela é imprimida por ordem alfabética.

Relativamente ao print das linhas da tabela, utilizamos um ciclo for para iterar sobre o array associativo e imprimir os valores de espaço ocupado seguidos do respectivo diretório. No final do ciclo for, recorremos ao comando sort juntamente com as devidas opções para ordenar a tabela segundo os argumentos passados pelo utilizador (verificados anteriormente). Também utilizamos o comando head com a flag -n para limitar o número de linhas impressas segundo o valor atribuído à variável limite.

Assim, a função print disponibiliza quatro opções de impressão, sendo que uma delas possibilita a utilização da ordenação alfabética e inversa ao mesmo tempo.

```

function print() {
    #verificação se passou algum diretório como argumento
    if [[ "${#space_array[@]}" -eq 0 ]]; then
        echo "Precisa de passar um diretório como argumento"
        exit 1
    fi

    #print final
    echo "SIZE NAME $(date +%Y%m%d) $@"
    if [[ $reverse -eq 1 ]]; then
        if [[ $sort_name -eq 1 ]]; then
            for val in "${!space_array[@]"; do
                echo "${space_array[$val]} $val"
            done | sort -k2,100r | head -n "$limite"
        else
            for val in "${!space_array[@]"; do
                echo "${space_array[$val]} $val"
            done | sort -k1,1n | head -n "$limite"
        fi
    else
        if [[ $sort_name -eq 1 ]]; then
            for val in "${!space_array[@]"; do
                echo "${space_array[$val]} $val"
            done | sort -k2,100 | head -n "$limite"
        else
            for val in "${!space_array[@]"; do
                echo "${space_array[$val]} $val"
            done | sort -k1,1nr | head -n "$limite"
        fi
    fi
}

```

fig. 6 | Função print responsável pela saída de dados

5) Main e verificações

Ao longo do script foram adicionadas verificações para garantir que não aconteciam erros/exceções e as variáveis não eram inicializadas com valores incorretos quando este fosse executado. Sempre que estas verificações falham, o programa imprime uma mensagem de erro e termina a sua execução (através do comando “exit 1”).

Na realidade a função **main()** é a primeira função a ser executada quando o programa é executado. Esta é responsável pelo bom funcionamento do código e pela correta ordem de execução do mesmo, uma vez que é ela que invoca a maior parte das outras funções juntamente com os argumentos introduzidos pelo utilizador.

```

function main() {
    input "$@"
    directories "$@"
    print "$@"
}

```

fig. 7| Função main, isto é, a primeira a ser chamada

Spacerate.sh

1) Argumentos de entrada

Para executar o script spacerate.sh, temos de passar como argumentos de entradas dois ficheiros que contém o output produzido pelo script spacecheck.sh. Este script permite a visualização da evolução do espaço ocupado no disco pelos ficheiros de um diretório.

Os ficheiros passados como argumento são as saídas produzidas pelo primeiro script e estão associados a diferentes períodos de tempo, permitindo verificar o espaço e os ficheiros que foram removidos ou adicionados numa dada diretoria.

De seguida, de modo a validar se os argumentos de entrada são dois ficheiros, desenvolvemos a função **files()**, que através de um ciclo for vai seleccionar cada um dos argumentos passados pelo terminal e proceder à sua verificação. Através de uma condição if dentro do ciclo for, analisamos se o argumento seleccionado é um ficheiro, e se corresponder, é chamada a função space.

```
function files() {  
    for i in "$@"; do  
        if [[ -f "$i" ]]; then  
            space "$i"  
        fi  
    done  
}
```

fig. 8 | Função para verificar se o argumento de entrada é um ficheiro

A função **input()** é responsável pela leitura dos argumentos introduzidos e verificar se existe algum argumento com o objetivo de alterar a ordem do output produzido pela função print. Utilizamos um ciclo while para processar os parâmetros passados pelo utilizador. Através do getopt, conseguimos verificar se os parâmetros de entrada são válidos. Os argumentos que são válidos são -a e -r, que têm exatamente a mesma função que no script anterior.

```
function input() {
    while getopts "ra" flag; do
        case $flag in
            r)
                reverse=1
                ;;
            a)
                sort_name=1
                ;;
            *)
                #caso seja passado um argumento não válido
                echo "Opção inválida! A sair..."
                exit 1
                ;;
        esac
    done
    shift $((OPTIND - 1))
}
```

fig. 9 | Função input com menos flags que o script spacecheck.sh

2) Variáveis inicializadas

Tal como no script anterior, foram inicializadas várias variáveis para assegurar o bom funcionamento do programa. Algumas variáveis têm um valor atribuído por defeito caso não sejam seleccionadas pelo utilizador. Este é o caso das variáveis **reverse**, **sort_name**, **arrayA_filled**, **arrayB_filled** e **spacedif**, que são inicializadas com o valor "0".

As variáveis **arrayA_filled** e **arrayB_filled** são usadas para garantir que o programa realiza as operações pretendidas na função space como verificar a existência de algum elemento dentro de um array.

```
#declaração de variaveis
declare reverse=0           #ordenação normal
declare sort_name=0        #ordenação default dos ficheiros
declare spacedif=0
declare arrayA_filled=0
declare arrayB_filled=0
```

fig. 10| Declaração das variáveis do script spacerate.sh

3) Cálculo do espaço ocupado

Para o cálculo final da evolução do espaço ocupado, desenvolvemos a função **space()**. A função **space** é invocada pela função **files**, depois de verificar se o argumento selecionado é um ficheiro. A função **space** é responsável pelo cálculo final do espaço atual ocupado pelos ficheiros, em diferentes períodos de execução.

Para armazenar os valores referentes a cada período, foi necessário declarar três arrays associativos **space_arrayA**, relativo ao ficheiro mais atual, **space_arrayB**, referente ao ficheiro mais antigo, **space_arrayfinal** para armazenar os resultados finais.

```
#declaração de arrays
declare -A space_arrayA
declare -A space_arrayB
declare -A space_arrayfinal
```

fig. 11 | Declaração dos arrays associativos do spacerate.sh

Inicialmente, na função **space()**, é criada uma variável local **file** e atribui-lhe o ficheiro inicialmente validado na função **files**. De seguida, é extraído as linhas do ficheiro a partir da segunda linha, através do comando **tail** juntamente com a flag **-n** seguido do valor “+2”, evitando assim a leitura e armazenamento do cabeçalho, caso este esteja presente no ficheiro. Isto só acontece se a condição do **if** se verificar, ou seja, a primeira linha do ficheiro começar com a palavra “**SIZE**”.

De modo a clarificar e a impedir a ocorrência de erros de execução, foi criado um ficheiro temporário **temp_file**, que permite que as linhas extraídas do ficheiro dado sejam redirecionadas para aqui.

O próximo passo é guardar as informações do ficheiro correspondente ao output mais recente no **space_arrayA** e as do ficheiro correspondente ao output mais antigo no **space_arrayB**. De modo a conseguir fazer esta separação, recorreremos a uma condição **if**. Como o primeiro array a ser lido é o atual, então decidimos que o **space_arrayA** deve ser o primeiro a ser preenchido. Por esta razão a condição **if** verifica se o **space_arrayA** foi preenchido e atribui à variável **arrayA_filled** o valor “1”. Este método permite que na próxima iteração do ciclo **for** da função **files**, como o valor atribuído ao **arrayA_filled** é “1”, então o programa começa a preencher o **space_arrayB** impedindo que ambos os arrays associativo sejam preenchidos na mesma iteração.

Os arrays vão ser lidos através do comando **read** que, usando o separador definido na variável **IFS**, isto é, estabelece um espaço em branco que permite dividir a linha em campos, lê e divide a linha do ficheiro em dois campos: o **size** e o **locat**. O **while** permite ler todas as linhas do campo até que não haja mais linhas para serem lidas.

Estes campos definidos por **size** e **locat** são armazenados nos respectivos arrays, dependendo se fazem parte do ficheiro atual ou do mais antigo, explicado anteriormente em cima.

Visto que o ficheiro temporário apenas servia para impedir problemas de execução associados à remoção da primeira linhas do ficheiro este deverá ser removido, através do comando `rm`.

Entrando agora na comparação entre os dois ficheiros. Em cada ficheiro passado como argumento, estão representados os espaços ocupados pelos ficheiros provenientes de um diretório por cada linha. O conjunto de todas as linhas do ficheiro, correspondem aos subdiretórios do diretório passado como argumento ao script “spacecheck.sh”. Aqui comparamos a existência e o tamanho dos ficheiros existentes no diretório, dos ficheiros-texto atual e antigo, verificado na função `files`.

Antes de proceder, verificamos sempre se o preenchimento correu como esperado e os arrays foram preenchidos. Este processo ocorre através das variáveis `arrayA_filled` e `arrayB_filled`.

A comparação está separada por duas partes, em que cada parte corresponde a um ciclo `for`.

O primeiro ciclo corresponde à comparação dos diretórios existentes no ficheiro-texto novo com os diretórios existentes no ficheiro-texto antigo. Assim, é atribuído a cada caminho a variável **first1** e o **tamanho** corresponde ao valor do caminho “\$first1” no `space_arrayB`. O principal objetivo é verificar se o caminho “\$first1” existe no ficheiro-texto antigo. Através da expressão de verificação “-n”, conseguimos averiguar se o tamanho do caminho no `space_arrayB` está ou não vazio. Se não estiver conseguimos perceber que é um ficheiro que se manteve e assim podemos calcular a evolução do espaço ocupado pelo mesmo. Para isso, atribuímos à variável `space_dif` o valor da diferença entre o tamanho do diretório no ficheiro-texto novo e o tamanho do diretório no ficheiro-texto antigo. Caso o diretório que existe no novo, não exista no antigo, devemos criar um novo nome para o diretório com a palavra “NEW”, a partir da variável **modify_first**.

Por fim, adicionamos o valor atribuído à variável `space_dif` ao `space_arrayfinal` associado com o respetivo nome do caminho alterado armazenado na variável `modify_first`.

O segundo ciclo corresponde à verificação da existência dos caminhos do `space_arrayB` no `space_arrayA`, ou seja, a possibilidade de haver algum diretório presente no ficheiro antigo, que tenha sido eliminado no ficheiro novo. Assim, vemos se o diretório existente no ficheiro antigo, não existe no ficheiro novo. Se isso se verificar, concluimos que o diretório foi removido e assim o tamanho ocupado por ele corresponde ao tamanho que ele apresenta no ficheiro antigo. Desta forma, devemos adicionar “REMOVED” ao nome do diretório, através da variável `modify_first` e armazenar os resultados no `space_arrayfinal`.

Concluindo esta função, temos um array associativo (`space_arrayfinal`) que tem a informação atualizada, relativa aos diretórios presentes nos dois ficheiros passados como argumentos.

```
function space() {
    local file="$1"
    local temp_file=$(mktemp) # Cria um arquivo temporário

    if [[ $(head -n 1 "$file") == "SIZE"* ]]; then
        tail -n +2 "$file" > "$temp_file"
    else
        # Caso a primeira linha não comece com "SIZE", apenas copie o arquivo original
        cp "$file" "$temp_file"
    fi

    if [ $arrayA_filled -eq 0 ]; then
        while IFS=' ' read -r size locat || [ -n "$size" ]; do
            space_arrayA["$locat"]=$size
        done < "$temp_file"
        arrayA_filled=1
    else
        while IFS=' ' read -r size locat || [ -n "$size" ]; do
            space_arrayB["$locat"]=$size
        done < "$temp_file"
        arrayB_filled=1
    fi

    rm "$temp_file" # Remove o arquivo temporário
}
```

fig. 12 | Função space que permite a manipulação dos ficheiros texto e preencher o space_arrayfinal

```
if [ "$arrayA_filled" -eq 1 ] && [ "$arrayB_filled" -eq 1 ]; then

    for first1 in "${!space_arrayA[@]}; do
        tamanho="${space_arrayA[$first1]}"
        if [[ -n "${space_arrayB[$first1]}" ]]; then
            tama="${space_arrayB[$first1]}"
            spacedif=$((tamanho-tama))
            space_arrayfinal[$first1]=$spacedif
        else
            modify_first="$first1 NEW"
            space_arrayfinal["$modify_first"]=$tamanho
        fi
    done

    for first2 in "${!space_arrayB[@]}; do
        if [[ -z "${space_arrayA[$first2]}" ]]; then
            spacedif=${space_arrayB[$first2]}
            modify_first="$first2 REMOVED"
            space_arrayfinal["$modify_first"]=$(( -1 * ${space_arrayB[$first2]} ))
        fi
    done

fi
}
```

fig. 13 | Continuação da função space

4) Print e formatação do output

Para finalizar o script “spacerate.sh”, realizamos a função **print()**. Começamos a função com a verificação do número de argumentos passados pelo utilizador e, de seguida, é realizada a impressão do cabeçalho da tabela. Por último a função imprime os valores armazenados no `space_arrayfinal`, ou seja, os resultados finais relativamente à evolução do espaço dos diretórios presentes nos ficheiros passados como argumentos. Esta impressão é feita segundo as opções de ordenação representadas pelos valores atribuídos às variáveis `reverse` e `sort_name`.

```
function print() {  
    if [[ "${#space_arrayfinal[@]}" -eq 0 ]]; then  
        echo "Precisa de passar dois ficheiros como argumento"  
        exit 1  
    fi  
  
    echo -e "SIZE NAME"  
    if [[ $reverse -eq 1 ]]; then  
        if [[ $sort_name -eq 1 ]]; then  
            for val in "${!space_arrayfinal[@]"; do  
                echo -e "${space_arrayfinal[$val]} $val"  
            done | sort -k2,100r  
        else  
            for val in "${!space_arrayfinal[@]"; do  
                echo -e "${space_arrayfinal[$val]} $val"  
            done | sort -k1,1n  
        fi  
    else  
        if [[ $sort_name -eq 1 ]]; then  
            for val in "${!space_arrayfinal[@]"; do  
                echo -e "${space_arrayfinal[$val]} $val"  
            done | sort -k2,100  
        else  
            for val in "${!space_arrayfinal[@]"; do  
                echo -e "${space_arrayfinal[$val]} $val"  
            done | sort -k1,1nr  
        fi  
    fi  
fi  
}
```

fig. 14 | Função print que valida e imprime os dados do `space_Arrayfinal`

VALIDAÇÃO DOS RESULTADOS OBTIDOS (ANÁLISE)

Com o objetivo de validar a nossa implementação, realizamos alguns testes. O script “spacecheck.sh” foi testado com o diretório “test_a1” disponibilizado no elearning. O script “spacerate.sh” foi testado com dois ficheiros (“spacecheck_novo.txt” e “spacecheck_antigo.txt”) que simulam o output do primeiro script, executado em períodos diferentes. Os ficheiros e os resultados apresentam-se a seguir:

1) Ficheiros teste

```
≡ spacecheck_novo.txt
1  6606 test_a1
2  6650 test_a1/test_a1
3  40368 test_a1/test_a1/aaa
4  9544 test_a1/test_a1/rrr
5  2000 test_a1/tomas
6  140 test_a1/edu
```

fig. 1 | Ficheiro teste atual criado para testar a eficiência dos scripts

```
≡ spacecheck_antigo.txt
1  SIZE NAME 20231108 test_a1
2  66808 test_a1
3  6608 test_a1/test_a1
4  4036 test_a1/test_a1/aaa
5  9544 test_a1/test_a1/rrr
6  7240 test_a1/test_a1/aaa/zzzz
```

fig. 2 | Ficheiro teste antigo criado para testar a eficiência dos scripts

2) Resultados do spacecheck

```
edu@edu:~/Desktop/S0/projeto/S0-Project/ProjetoS0$ ./spacecheck.sh -r test_a1
SIZE NAME 20231110 -r test_a1
7240 test_a1/test_a1/aaa/zzzz
9544 test_a1/test_a1/rrr
40368 test_a1/test_a1/aaa
66508 test_a1/test_a1
66806 test_a1
```

fig. 3 | Teste realizado para a flag -r (ordem inversa de tamanho)

```
edu@edu:~/Desktop/S0/projeto/S0-Project/ProjetoS0$ ./spacecheck.sh -a test_a1
SIZE NAME 20231110 -a test_a1
66806 test_a1
66508 test_a1/test_a1
40368 test_a1/test_a1/aaa
7240 test_a1/test_a1/aaa/zzzz
9544 test_a1/test_a1/rrr
```

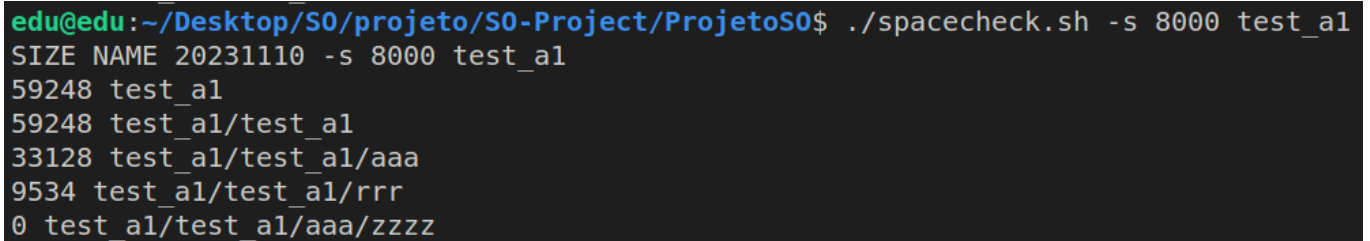
fig. 4 | Teste realizado para a flag -a (ordenado por nome)

```
edu@edu:~/Desktop/S0/projeto/S0-Project/ProjetoS0$ ./spacecheck.sh -a -r test_a1
SIZE NAME 20231110 -a -r test_a1
9544 test_a1/test_a1/rrr
7240 test_a1/test_a1/aaa/zzzz
40368 test_a1/test_a1/aaa
66508 test_a1/test_a1
66806 test_a1
```

fig. 5 | Teste realizado para a flag -a e -r em simultâneo

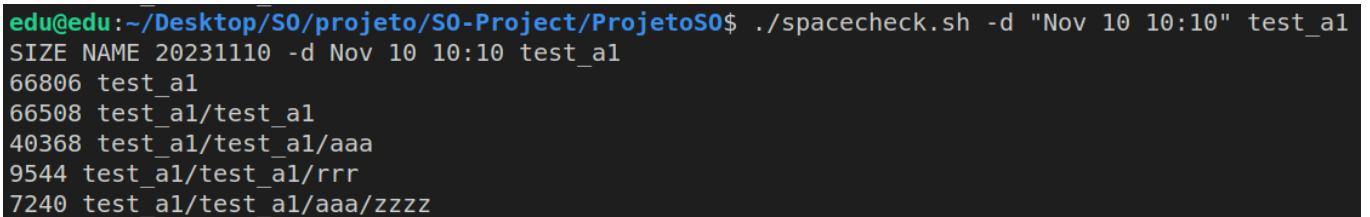
```
edu@edu:~/Desktop/S0/projeto/S0-Project/ProjetoS0$ ./spacecheck.sh -l 3 test_a1
SIZE NAME 20231110 -l 3 test_a1
66806 test_a1
66508 test_a1/test_a1
40368 test_a1/test_a1/aaa
```

fig. 6 | Teste realizado para a flag -l (número máximo de linhas apresentadas)



```
edu@edu:~/Desktop/S0/projeto/S0-Project/ProjetoS0$ ./spacecheck.sh -s 8000 test_a1
SIZE NAME 20231110 -s 8000 test_a1
59248 test_a1
59248 test_a1/test_a1
33128 test_a1/test_a1/aaa
9534 test_a1/test_a1/rrr
0 test_a1/test_a1/aaa/zzzz
```

fig. 7 | Teste de tamanho mínimo e todos os que tiverem abaixo revelam o valor 0



```
edu@edu:~/Desktop/S0/projeto/S0-Project/ProjetoS0$ ./spacecheck.sh -d "Nov 10 10:10" test_a1
SIZE NAME 20231110 -d Nov 10 10:10 test_a1
66806 test_a1
66508 test_a1/test_a1
40368 test_a1/test_a1/aaa
9544 test_a1/test_a1/rrr
7240 test_a1/test_a1/aaa/zzzz
```

fig. 8 | Teste realizado para a data máxima de criação/modificação

3) Resultados do spacerate

```
edu@edu:~/Desktop/S0/projeto/S0-Project/ProjetoS0$ ./spacerate.sh spacecheck_novo.txt spacecheck_antigo.txt
SIZE NAME
36332 test_al/test_al/aaa
2000 test_al/tomas NEW
140 test_al/edu NEW
42 test_al/test_al
0 test_al/test_al/rrr
-7240 test_al/test_al/aaa/zzzz REMOVED
-60202 test_al
```

fig. 9 | Comparação dos dois ficheiros texto. Visualização da evolução do tamanho e dos ficheiros

```
edu@edu:~/Desktop/S0/projeto/S0-Project/ProjetoS0$ ./spacerate.sh -a spacecheck_novo.txt spacecheck_antigo.txt
SIZE NAME
-60202 test_al
140 test_al/edu NEW
42 test_al/test_al
36332 test_al/test_al/aaa
-7240 test_al/test_al/aaa/zzzz REMOVED
0 test_al/test_al/rrr
2000 test_al/tomas NEW
```

fig. 10 | Comparação dos ficheiros, mas com ordem alterada por nome

```
edu@edu:~/Desktop/S0/projeto/S0-Project/ProjetoS0$ ./spacerate.sh -r spacecheck_novo.txt spacecheck_antigo.txt
SIZE NAME
-60202 test_al
-7240 test_al/test_al/aaa/zzzz REMOVED
0 test_al/test_al/rrr
42 test_al/test_al
140 test_al/edu NEW
2000 test_al/tomas NEW
36332 test_al/test_al/aaa
```

fig. 11 | Comparação dos ficheiros, mas por ordem inversa de tamanho

```
edu@edu:~/Desktop/S0/projeto/S0-Project/ProjetoS0$ ./spacerate.sh -r -a spacecheck_novo.txt spacecheck_antigo.txt
SIZE NAME
2000 test_al/tomas NEW
0 test_al/test_al/rrr
-7240 test_al/test_al/aaa/zzzz REMOVED
36332 test_al/test_al/aaa
42 test_al/test_al
140 test_al/edu NEW
-60202 test_al
```

fig. 12 | Comparação dos ficheiros, mas com ordem alterada por nome e reverso de tamanho

4) Erros e Avisos

```
edu@edu:~/Desktop/S0/projeto/S0-Project/ProjetoS0$ ./spacecheck.sh -d "nov 10 25:10" test_a1
Data inválida
```

fig 13 | Erro relativo à introdução de uma data inválida

```
edu@edu:~/Desktop/S0/projeto/S0-Project/ProjetoS0$ ./spacecheck.sh -s -1 test_a1
Número inserido inválido
```

fig 14 | Erro relativo à introdução de um número que não é inteiro positivo

```
edu@edu:~/Desktop/S0/projeto/S0-Project/ProjetoS0$ ./spacecheck.sh -l -1 test_a1
Número inserido inválido
```

fig 15 | Erro relativo à introdução de um número que não é positivo

```
edu@edu:~/Desktop/S0/projeto/S0-Project/ProjetoS0$ ./spacecheck.sh -i test_a1
./spacecheck.sh: opção ilegal -- i
Opção inválida! A sair...
```

fig 16 | Erro relativo à introdução de uma flag que não está criada no spacecheck(mesmo erro no spacerate)

```
edu@edu:~/Desktop/S0/projeto/S0-Project/ProjetoS0$ ./spacecheck.sh
Precisa de passar um diretório como argumento
```

fig 17 | Erro relativo à introdução um argumento que não é um diretório válido

```
edu@edu:~/Desktop/S0/projeto/S0-Project/ProjetoS0$ ./spacerate.sh spacecheck_novo.txt
Precisa de passar dois ficheiros como argumento
```

fig 18 | Erro relativo à introdução de dois argumentos que não são dois ficheiros de texto



CONCLUSÃO

Em conclusão, a realização deste trabalho permitiu colocar em prática o que foi lecionado nas aulas práticas.

O facto de termos conseguido obter resultados bastante aceitáveis permite-nos perceber que alcançámos todos os objetivos estipulados para este projeto. Foi bastante enriquecedor a realização deste trabalho e, para além disso, conseguimos melhorar as nossas capacidades de trabalho de grupo e a nossa capacidade de conseguir encontrar e recolher informação relevante para a realização do projeto.

BIBLIOGRAFIA

- <https://stackoverflow.com/questions/273664/best-awk-commands>
- <https://stackoverflow.com/questions/16661982/check-folder-size-in-bash>
- <https://man7.org/linux/man-pages/man3/getopt.3.html>
- <https://www.baeldung.com/linux/check-file-type-exists-in-directory>
- <https://www.geeksforgeeks.org/find-command-in-linux-with-examples/>
- <https://www.linuxtechi.com/stat-command-examples-in-linux/>
- <https://levelup.gitconnected.com/change-permissions-of-a-file-in-bash-71651c4fc527>
- <https://stackoverflow.com/questions/58423265/checking-file-permissions-in-linux>
- <https://stackoverflow.com/questions/65115761/dynamically-creating-associative-arrays-in-bash>
- <https://stackoverflow.com/questions/44122714/how-to-execute-bash-script-in-same-shell>
- <https://pt.stackoverflow.com/questions/95760/receber-linhas-de-arquivo-e-trat%C3%A1-las-bash>
- <https://phoenixnap.com/kb/bash-read>
- <https://ioflood.com/blog/bash-read-man-page-using-the-linux-read-command/>
- <https://stackoverflow.com/questions/62889930/shell-script-to-perform-column-level-data-validation>
- <https://www.datacamp.com/tutorial/shell-commands-data-scientist>
- <https://www.pluralsight.com/resources/blog/cloud/conditions-in-bash-scripting-if-statements>