VISUALIZAÇÃO DO ESPAÇO DE DIRETORIAS

Sistemas Operativos

Tomás Costa - 89016
João Carvalho - 89059

Índice

Introdução

Na unidade curricular de Sistemas Operativos foi pedido aos alunos como primeiro projeto prático que escrevessem dois scripts, denominados totalspace.sh e nespace.sh.

Através do script totalspace.sh deveria ser possível observar o tamanho de todos os ficheiros de uma diretoria passada e das suas subdiretorias, este script deveria ainda ser capaz de ter opções especiais passadas no terminal.

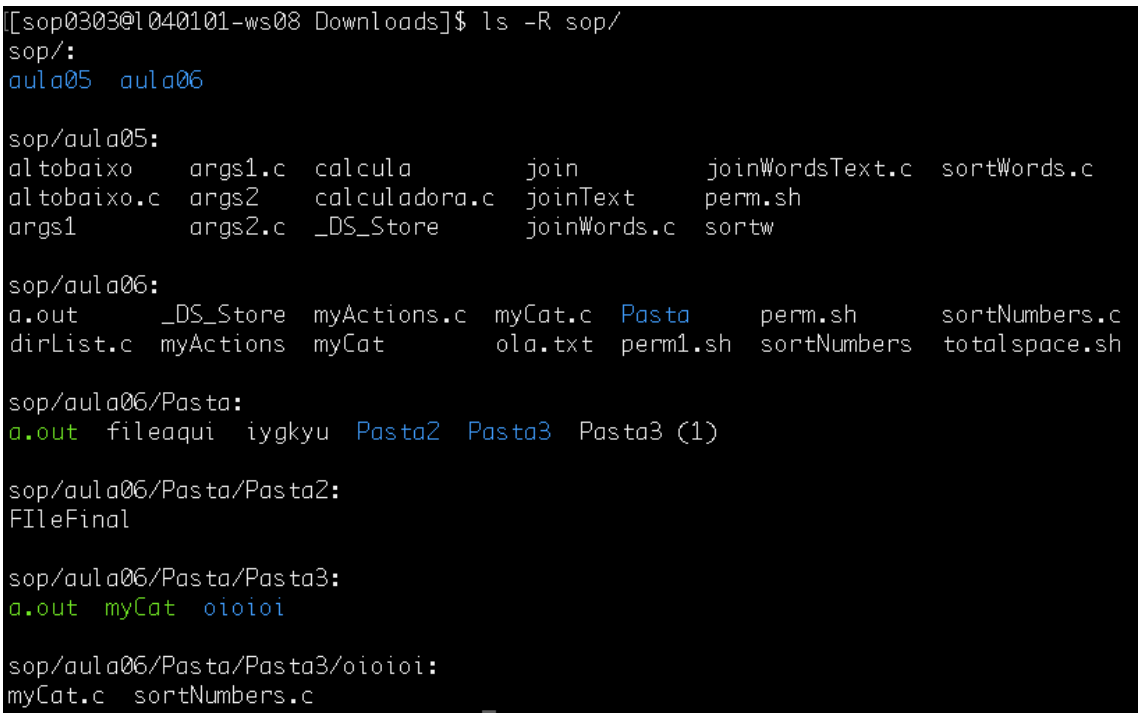
O script nespace.sh e semelhante ao totalspace.sh, no entanto contem mais uma opção especial -e que acrescenta uma funcionalidade de indicação de uma lista de ficheiros considerados essenciais e que por isso se considera não deverem ser contabilizados ao determinar o espaço ocupado em disco.

Este trabalho foi realizado usando BASH.

Antes de Implementar

Antes de começarmos a escrever código, escrevemos uma primeira abordagem ao problema no qual tínhamos 3 objetivos claros: fazer uma função que percorresse todos os ficheiros da diretoria passada, fazer verificação de ficheiro ou diretoria, e depois chamar recursivamente a função quando fosse uma diretoria para listar tudo.

E ainda antes de começarmos a escrever o código lemos cuidadosamente o guião para percebermos que caminhos devíamos tomar e de acordo com as opções criamos uma pasta de testes chamada “sop” com alguns ficheiros que nos permitisse correr testes



Primeira Abordagem

Numa primeira abordagem criamos a função recursiva que verificasse se era ficheiro ou diretório, como tínhamos definido em papel.
Para isso percorremos todo os ficheiros do diretório passado como argumento:

```
for file in $dirMae/*; do
    if [ -d ${file} ] ;
    then
        dirMae=$file
        myfunc1
    elif [ -f ${file} ] ;
    then #Ir buscar tamanhos total e dps devolver a funçao e guardar
        tamanho=$(stat $file | head -2 | tail -1 | awk '{print $2}')
        tamanhoTotal=$(($tamanhoTotal+$tamanho))
    fi
done
```

Depois calculamos o tamanho desse ficheiro usando este comando e incrementamos no tamanho total dessa diretoria, como demonstrado:

```
tamanho=$(stat $file | head -2 | tail -1 | awk '{print $2}')
tamanhoTotal=$(($tamanhoTotal+$tamanho))
```

Esta abordagem trouxe-nos alguns problemas pois caso estivéssemos a ler ficheiros e entrássemos numa diretoria antes de lermos todos os seus ficheiros não iria passar bem o tamanho da diretoria para trás

Segunda Abordagem

Numa segunda abordagem tentamos listar primeiro os ficheiros e depois as diretorias para conseguirmos passar corretamente o valor do tamanho, mas esta tentativa não teve sucesso pois deparamo-nos com vários erros de pensamento relativo a recursividade.

Íamos faze-lo através deste ciclo for, resumidamente dando grep -v as diretorias e só depois grep as diretorias

```
for file in $(ls -l "$dirMae" | grep -v "^d" | awk '{for(i=9; i<NF; ++i) printf "%s ", $i; if($i != " ") printf "%s", $i; print ""}'
&& ls -l "$dirMae" | grep "^d" | awk '{for(i=9; i<NF; ++i) printf "%s ", $i; if($i != " ") printf "%s", $i; print ""}')
```

Nesta abordagem já inicializamos os 4 arrays que iriamos usar, a nossa ideia seria que apesar de não serem arrays associativos iriam estar associados pelo índice, logo tendo um índice tínhamos toda informação que pretendíamos.
Como ainda não tínhamos integrado o getOpts, usamos um ciclo para percorrer todos os argumentos passados e atualizava-mos o ultimo para ser o diretório passado na função.

```
data=()
tamanhos=()
nome=()
tipo=()

tamanhoTotal=0
i=0
for i in $@ ; do
    dirMae=$i
done
```

Também tivemos muitos problemas em passar o tamanho recursivamente para as diretorias de trás, pelo que experimentamos muita coisa.

Infelizmente não chegamos a nenhuma conclusão durante alguns dias pelo que decidimos refazer o código usando arrays associativos para a informação estar melhor organizada.

Terceira Abordagem (Final)

Nesta terceira abordagem foi onde fizemos mais progresso, pois ao reescrevermos o código conseguimos perceber melhor as iterações pelo que conseguimos resolver os nossos problemas de tamanho das diretorias

Criamos 2 arrays associativos, um para guardar as diretorias e respetivos tamanhos e outro para guardar os ficheiros e os seus tamanhos. Criamos também 2 arrays normais, um para guardar as diretorias e outro para guardar todos os tamanhos dessas diretorias.

```
declare -A dicionarioFicheiro
declare -A dicionarioMae
arrayTamanhos=()
arrayDiretorias=("$1")
```

Leitura para os arrays

O nosso código começa com a função "guardarFiles()" que irá percorrer as subdiretorias e ficheiros de uma diretoria passada como argumento ($1). Se for um ficheiro, vamos encontrar o seu tamanho e a sua última data de acesso, depois tratamos das condições para as combinações dos argumentos -n e -d.

```
tamanhoTotal=0
for file in "$1"/* ; do
    if [ -f $file ] ;
    then
            tamanhoLocal=$(stat $file | head -2 | tail -1 | awk '{print $2}')
            dataFile=$(stat $file | head -5 | tail -1 | awk '{print $2, $3}')
            dataFile=$(date -d $dataFile "+%Y%m%d%H%M")
```

Caso -n

Considerámos apenas os ficheiros que têm um nome que contêm a expressão regex passada como argumento do -n.

```
elif [ $n_flag -eq 1 ]; then
    if [[ $file =~ $n_arg ]];then
        tamanhoTotal=$((tamanhoTotal + tamanhoLocal));
        dicionarioFicheiro[$file]=$tamanhoLocal
    fi
```

Caso -d

Considerámos apenas os ficheiros cuja data de acesso não ultrapasse a data máxima passada como argumento do -d

```
elif [ $d_flag -eq 1 ]; then
    if  [ $dataFile -le $d_arg ]; then
        tamanhoTotal=$((tamanhoTotal + tamanhoLocal));
        dicionarioFicheiro[$file]=$tamanhoLocal
    fi
```

Caso -n e -d

Considerámos apenas os ficheiros que correspondem as duas condições anteriores em simultâneo

Esta função e a primeira a aparecer porque assim fazemos menos verificações.

```
if [ $n_flag -eq 1 -a $d_flag -eq 1 ]; then
    if [[ $dataFile -le $d_arg && $file =~ $n_arg ]]; then
        tamanhoTotal=$((tamanhoTotal + tamanhoLocal));
        dicionarioFicheiro[$file]=$tamanhoLocal
    fi
```

Sem opções especiais vamos repetir os dois comandos sem nenhuma verificação.

Em todos estes casos pegamos no tamanho do ficheiro($tamanhoLocal) e adicionamos ao tamanhoTotal, que é uma variável inicializada a 0 no inicio da função e que guarda o tamanho da diretoria $1, além disso guardámos esse tamanhoLocal num array associativo ($dicionarioFicheiro) que tem como chave o nome do ficheiro($file) e valor, o tamanho desse ficheiro ($tamanhoLocal).

Caso o $file seja uma diretoria, guardámos o $file num array (arrayDiretorias), que guarda todas as diretorias, e guardámos o tamanho dessa diretoria ($tamanhoTotal) noutro array(arrayTamanhos) que guarda todos os tamanhos das diretorias, temos um array com as diretorias todas cujos tamanhos se encontram no arrayTamanhos com o mesmo índice.

```
elif [ -d "${file}" ] ;then
    arrayDiretorias+=($file)
    arrayTamanhos+=($tamanhoTotal)
    guardarFiles "${file}"
fi
```

Display dos valores

Para darmos display aos valores chamamos a função showVals()
Nesta função tratamos do -l para só considerar os maiores ficheiros de cada diretoria. Neste tratamento percorremos as diretorias e guardamos num array todos os ficheiros e subdiretorias dessa diretoria

```
function showVals() {
    if [ $l_flag -eq 1 ]; then
        for i in "${!dicionarioMae[@]}"
        do
            path=0;
            tamanhoLocal_b=0;
            arrayTamanhosFinal=()
            arrayComTudo=($( ls -l $i | sort -nr | awk '{print $9}'))
            for j in "${arrayComTudo[@]}"
            do
                path="$i/$j"
                if [ -f $path ]; then
                    tamanhoLocal=$(ls -l ${path} | awk '{print $5}')
                    dataFile=$(stat $file | head -5 | tail -1 | awk '{print $2, $3}')
                    dataFile=$(date -d $dataFile "+%Y%m%d%H%M")

                    if [ $n_flag -eq 1 -a $d_flag -eq 1 ]; then
                        if [[ $dataFile -le $d_arg && $path =~ $n_arg ]]; then
                            arrayTamanhosFinal+=( $tamanhoLocal )
                        fi
                    elif [ $n_flag -eq 1 ]; then
                        if [[ $path =~ $n_arg ]];then
                            arrayTamanhosFinal+=( $tamanhoLocal )
                        fi
                    elif [ $d_flag -eq 1 ]; then
                        if [ $dataFile -le $d_arg ]; then
                            arrayTamanhosFinal+=( $tamanhoLocal )
                        fi
                    else
                        arrayTamanhosFinal+=( $tamanhoLocal )
                    fi
                fi
            done
            arrayTamanhosFinal=( $( printf "%s\n" "${arrayTamanhosFinal[@]}" | sort -nr | head -${l_arg} ) )
            for k in "${arrayTamanhosFinal[@]}"
            do
                tamanhoLocal_b=$((tamanhoLocal_b + k));
            done
```

A data e guardada num formato “ano:mês:dia:horas:minutos” (sem :) pelo que se torna fácil de comparar datas porque se tratarmos como um inteiro, qualquer data mais “recente” vai ser um inteiro maior. Logo se o argumento passado for menor que a data dos ficheiros e porque nos podemos guardar no array.

Depois percorremos todos esses ficheiros e subdiretorias e com a variável “path” conseguimos ir buscar o tamanho e data desse ficheiro, vamos fazer as verificações de opções extra

Quando acabamos de percorrer o arrayComTudo, colocamos no array associativo de diretorias e tamanhos os seus valores respetivos, mas apenas vamos buscar os l_arg maiores, que e um inteiro fornecido na chamada da função.

```
path=0;
tamanhoLocal_b=0;
arrayTamanhosFinal=()
arrayComTudo=($( ls -l $i | sort -nr | awk '{print $9}'))
for j in "${arrayComTudo[@]}"
do
    # Se $j for um ficheiro
    path="$i/$j"
    if [ -f $path ]; then
        tamanhoLocal=$(ls -l ${path} | awk '{print $5}')
        dataFile=$(stat $file | head -5 | tail -1 | awk '{print $2, $3}')
        dataFile=$(date -d $dataFile "+%Y%m%d%H%M")
```

Depois verificamos o -L (caso haja -l e -L esta coberto pela verificação) pelo que basta ir buscar os maiores ficheiros com o inteiro fornecido

```
            dicionarioMae[$i]=$tamanhoLocal_b
            echo ${dicionarioMae[$i]} $i
        done | sort $s
    fi

    if [ $L_flag -eq 1 ];then
        for k in "${!dicionarioFicheiro[@]}"
        do
            echo ${dicionarioFicheiro["$k"]} $k
        done | sort -nr -k1 | head -${L_arg}
    else

        for i in "${!dicionarioMae[@]}"
        do
            final_space=0;
            diretorios=($( ls -lhR $i | grep '/' ))

            for j in "${diretorios[@]}"
            do
                j=${j%?}
                final_space=$((dicionarioMae[$j] + $final_space))
            done
            if [ $l_flag -eq 0 ];then

                echo $final_space $i

            fi
        done  | sort $s
    fi
```

Senão vamos imprimir todas as diretorias e o respetivo tamanho, usamos o comando ls -lR para recursivamente mostrar as diretorias e subdiretorias, no fim imprimimos as diretorias e tamanhos e usamos | para depois ficar sorted.

Tratamento de Opções

Para o getOpts começamos por declarar todas as variáveis flag para cada argumento que servem como para sinalizar se foi chamado (flag = 1) ou não (flag = 0), pelo que começam todas a 0, inicializamos também os argumentos das que requerem argumento a 0.

```
IFS=$'\n'
# Opções
    a_flag=0
    d_flag=0
    l_flag=0
    L_flag=0
    n_flag=0
    r_flag=0

    d_arg=0
    l_arg=0
    L_arg=0
    n_arg=0
```

Depois fazemos o tratamento das opções da optstring, este tratamento e o getOpts foram uma adaptação do que fizemos em C nas aulas praticas e pelo manual de getOpts:

```
while getopts ":l:L:n:rad:" opt; do
    case ${opt} in
        n) # Procura por nome dos ficheiros
            verificarDobro "$n_flag"
            n_arg=${OPTARG}
            verificarArgs "${n_arg}"
            n_flag=1
            ;;

        l) # Maiores ficheiros em cada diretoria
            l_arg=${OPTARG}
            verificarDobro $l_flag
            l_flag=1
            verificarArgs $l_arg
            ;;

        d) # Data máxima
            verificarDobro $d_flag
            d_arg=${OPTARG}
            verificarArgs "${d_arg}"
            d_flag=1
            ;;

        L) # Maiores ficheiros em todas as diretoria
            verificarDobro $L_flag
            L_arg=${OPTARG}
            verificarArgs "${L_arg}"
            L_flag=1
            ;;
```

```
        r) # ordenar os valores
            verificarDobro $r_flag
            r_flag=1
            ;;

        a) # ordenar alfabeticamente
            verificarDobro $a_flag
            a_flag=1
            ;;
        *)
                    usoerro
                    ;;
    esac
done
```

No final do getOpts fazemos um shift, para atualizarmos o índice da diretoria

```
shift $(($OPTIND - 1))
```

Para cada opção vamos verificar se essa opção já foi chamada e verificamos se tem argumento, em caso de incumprimento e mostrada na terminal a mensagem de erro, explicada mais a frente.

```
if [[ $# -eq 0 ]]; then
    usoerro
fi

for i in $@; do
    if ! [[ -d $i ]]; then
        usoerro
    fi
done
```

Funções de verificação

Criamos uma função chamada usoerro() que imprime no terminal o uso correto da função que e chamada de um modo não autorizado.

```
function usoerro(){
    echo "Incorrect Usage of arguemnts."
    echo "Usage: ./totalspace.sh <Options> <Directories>"
    echo "   Examples:"
    echo "         Option: -l k, choose int_k biggest files from each directory"
    echo "         Option: -L k, choose int_k biggest files in all directories"
    echo "         Option: -n <.*sh>, only files that match regex expression"
    echo "         Option: -r, sort in reverse"
    echo "         Option: -a, sort alphabetically"
    echo "         Option: -d 'Sep 10 2018 10:00', acesso data maximo ao file"
    echo "Warning! Options -l and -L can't be used simultaneosly, all other combinations allowed"
    echo "Warning! Options [ -L , -l , -n , -d ] require arguments."
    exit 1 #Terminar e indicar erro
}
```

Criamos também duas funções para verificar os argumentos do getOpts e as suas, uma para verificar se os que requerem argumentos os têm (se a seguir a opção não vier outra opção) e outra para verificar se já usamos aquele argumento (se a flag já estiver a 1)

```
function verificarArgs(){
    if [[ $1 == -* ]]
    then
        echo "Error: $1 can't be an option, it has to be an argument"
        usoerro
    fi
}

function verificarDobro(){
    if [[ $1 -eq 1 ]]
    then
        echo "Error: $opt has been used already"
        usoerro
    fi
}
```

Verificações extra das opções

Antes de corrermos o programa vamos ter que verificar algumas coisas, como:

Se não passaram nenhuma diretoria,
Se o argumento da -l e um inteiro,
Se o argumento da -L e um inteiro,
Se não estamos a usar -l e -L que não e permitido,
Se o argumento da -n e uma expressão regex válida,
Se o data e passado com um argumento válido,

```
#verificar se e inteiro
if [[ $l_flag -eq 1 ]]; then
    if ! [[ $l_arg =~ ^[0-9]+$ ]]; then
        usoerro
    fi
fi

#verificar se e inteiro
if [[ $L_flag -eq 1 ]]; then
    if ! [[ $L_arg =~ ^[0-9]+$ ]]; then
        usoerro
    fi
fi

#Verificar ambos -l e -L
if [[ $l_flag -eq 1 && $L_flag -eq 1 ]]; then
    usoerro
fi

if [[ $n_flag -eq 1 ]]; then
    if ! [[ $n_arg =~ ^[0-9A-Za-z*.$_]+$ ]]; then
        usoerro
    fi
fi
```

Ordenação dos tamanhos

Nos fazemos um tratamento para ordenar caso sejam passadas as opções -a (alfabeticamente) e -r (ordem inversa).
Para tal atualizamos uma variável chamada s com o argumento especial do sort e depois só precisamos de o chamar em cima com (sort $s).

```
if [ $r_flag -eq 1 -a $a_flag -eq 0 ]; then
    s="-nr"
elif [ $a_flag -eq 1 -a $r_flag -eq 0 ]; then
    s="-k2"
elif [ $r_flag -eq 1 -a $a_flag -eq 1 ]; then
    s="-k2r"
else
    s="-n"
fi
```

Chamada de funções

Na nossa “main” declaramos os arrays supracitados, vamos fazer a chamada a função guardarFiles() e como a diretoria passada também vai apresentar um tamanho, basta que usemos o tamanho total calculado e adicionemos ao array de tamanhos depois da chamada.

Para depois associarmos os tamanhos a respetiva diretoria fazemos um ciclo for que percorre as diretorias e atribuímos ao value (essa diretoria) o tamanho do array de tamanhos.

```
guardarFiles $1
arrayTamanhos+=("${tamanhoTotal}")

count=0;
for i in "${arrayDiretorias[@]}"
do
    dicionarioMae["${i}"]="${arrayTamanhos[$count]}"
    ((count++));
done

showVals
```

No final e chamada a função showVals() e imprime os valores na terminal

Segundo Script

A ideia e implementação do segundo script e bastante fácil visto que já temos o código do totalspace.sh e eles são semelhantes. Para implementar este script foi necessário adicionarmos uma opção extra no getOpts que recebe como argumento um ficheiro texto, e também temos que verificar se o argumento passado e um ficheiro

```
e)
    verificarDobro $e_flag
    e_arg=${OPTARG}
    verificarArgs
    e_flag=1
    ;;
```

```
if [[ $e_flag -eq 1 ]]; then
    if ! [[ -f $e_arg ]]; then
        usoerro
    fi
fi
```

O tratamento de dados resume-se a ler as linhas do ficheiro para um array, visto que os ficheiros escondidos estão 1 por linha, depois de ler temos que fazer uma verificação antes de considerarmos o ficheiro para ver se esta escondido ou não

```
read -d '' -r -a filesEscondidos < $e_arg

echo "Estes files estao escondidos: " "${filesEscondidos[@]}"
```

```
contemElemento "${file}" "${filesEscondidos[@]}"
#se return for 1 e porque nao pertence
if [ -f $file && $? -eq 1 ] ;
```

Para isso adaptei uma função simples (StackOverflow) para ver se um elemento pertence a um array ou não, e essa função devolve 1 se não pertencer (e então podemos avançar) ou 0 se pertencer, nos podemos aceder ao valor deste retorno com $? dentro da condição if.

```
function contemElemento() {
    local e match="$1"
    shift

    for e;
        do [[ "$e" == "$match" ]] && return 0
    done

    return 1
}
```

Conclusões

Na ultima versão do trabalho não conseguimos por a verificação de diretórios e ficheiros sem acesso a funcionar, apesar de acharmos estar perto da solução, visto que caso o ficheiro ou diretoria não tenha permissões de read, não podemos fazer o ls ai, logo basta barrarmos com um if [[-r ficheiro]], sendo que assim só somamos valores quando o file e readable, senão atribuímos tamanhoTotal dessa diretoria a NA, e depois a percorrer diretorias caso algum tamanho seja NA, também essa diretoria assume valor de NA.

E importante mencionar que o código não foi desenvolvido com o intuito de obter uma melhor otimização do mesmo, mas sim com o principal objetivo do funcionamento de todas as funcionalidades requeridas.

Seguem-se alguns exemplos de utilização

```
[sop0303@l040101-ws08 Downloads]$ ./totalspacetest.sh -n ".*sh" sop
0 sop/aula06/Pasta/Pasta2
25 sop/aula05
1090 sop/aula06
1090 sop/aula06/Pasta
1090 sop/aula06/Pasta/Pasta3
1090 sop/aula06/Pasta/Pasta3/oioioi
1115 sop
[sop0303@l040101-ws08 Downloads]$ ./totalspacetest.sh -n ".*sh" -L 2 sop
1040 sop/aula06/totalspace.sh
25 sop/aula06/perm.sh
[sop0303@l040101-ws08 Downloads]$ ./totalspacetest.sh -n ".*sh" -l 1 sop
0 sop
0 sop/aula06/Pasta
0 sop/aula06/Pasta/Pasta2
0 sop/aula06/Pasta/Pasta3
0 sop/aula06/Pasta/Pasta3/oioioi
25 sop/aula05
1040 sop/aula06
[sop0303@l040101-ws08 Downloads]$ ./totalspacetest.sh -n ".*sh" -d "Nov 11 11:00" sop
0 sop
0 sop/aula05
0 sop/aula06
0 sop/aula06/Pasta
0 sop/aula06/Pasta/Pasta2
0 sop/aula06/Pasta/Pasta3
0 sop/aula06/Pasta/Pasta3/oioioi
[sop0303@l040101-ws08 Downloads]$ ./totalspacetest.sh -a -l 3 sop
0 sop
25520 sop/aula05
25656 sop/aula06
8598 sop/aula06/Pasta
12 sop/aula06/Pasta/Pasta2
17120 sop/aula06/Pasta/Pasta3
1980 sop/aula06/Pasta/Pasta3/oioioi
[sop0303@l040101-ws08 Downloads]$ ./totalspacetest.sh -a -r -l 3 sop
1980 sop/aula06/Pasta/Pasta3/oioioi
17120 sop/aula06/Pasta/Pasta3
12 sop/aula06/Pasta/Pasta2
8598 sop/aula06/Pasta
25656 sop/aula06
25520 sop/aula05
0 sop
```

Bibliografia

https://stackoverflow.com/questions/16483119/an-example-of-how-to-use-getopts-in-bash

http://wiki.bash-hackers.org/syntax/shellvars

http://ahmed.amayem.com/bash-recursion-examples-and-experiments-with-local-variables/

https://stackoverflow.com/questions/5168071/list-sub-directories-with-ls

Comando ***man*** no terminal (visto que inclui vários use-cases de comandos BASH)

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

https://unix.stackexchange.com/questions/384036/script-that-will-read-5-numbers-and-then-sort-from-highest-to-lowest

https://stackoverflow.com/questions/3685970/check-if-a-bash-array-contains-a-value