

## **Ferramentas De Descoberta do Espaço Ocupado em Disco**

**Guilherme Moura (64770)**

**André Ribas (70150)**

**23 de novembro de 2018**



**Trabalho efetuado no âmbito da disciplina de**  
**Sistemas Operativos**  
**da Licenciatura em Engenharia Informática**



## Introdução

O objetivo deste trabalho é o desenvolvimento de um script em bash para descobrir o espaço ocupado em disco por ficheiros com determinadas propriedades e que podem ser candidatos a ser apagados. Deverá permitir ver o espaço ocupado em disco por todos os ficheiros e eliminar aqueles que não são considerados essenciais.

Este trabalho vai estar dividido em dois scripts em bash, o primeiro será o script `totalspace.sh` que irá permitir ver a visualização do espaço ocupado pelos ficheiros selecionados. O segundo script `nespace.sh` será idêntico ao primeiro mas acrescenta uma funcionalidade de indicação de uma lista de ficheiros considerados essenciais e que por isso se considera que não devem ser contabilizados para determinar o espaço ocupado em disco onde a ideia será que estes ficheiros nunca podem ser apagados e, logo, a atenção sobre o espaço utilizado deve ser focado em outros ficheiros.

Este trabalho terá de ter certos parâmetros pré-definidos de maneira a que se possam visualizar todas as opções definidas pelo professor assim como a manipulação de dados.

## Desenvolvimento do trabalho

Nos scripts totalspace.sh e nespace.sh começamos a realizar a solução do nosso problema com a verificação dos nossos argumentos onde todos os argumentos foram implementados.

```
#Verificação de argumentos
while getopts n:l:L:d:ra opt; do
  case "$opt" in
    n ) #Seleccao atraves do nome do ficheiro
        opn="$OPTARG"
        echo "-n: $OPTARG"
        ;;
    l ) #Quantos entre os maiores de cada directoria devem ser considerados
        if [[ "$opL" -ne "i" ]]; then
            echo "Não pode usar -l e -L ao mesmo tempo"
            exit 1
        fi
        if [[ "$OPTARG" =~ ^[0-9]+$ ]]; then
            opl="$OPTARG"
        else
            echo "Opção -l "$OPTARG" não é um número"
        fi
        echo "-l: $OPTARG"
        ;;
    L ) #Quantos entre os maiores de TODAS as directorias devem ser considerados
        if [[ "$opL" -ne "i" ]]; then
            echo "Não pode usar -l e -L ao mesmo tempo"
            exit 1
        fi
        if [[ "$OPTARG" =~ ^[0-9]+$ ]]; then
            opl="$OPTARG"
        else
            echo "Opção -L "$OPTARG" não é um número"
        fi
        echo "-L: $OPTARG"
        ;;
  esac
done
shift $(( OPTIND - 1 ))
```

```
;;
d ) #Especificacao da data maxima de acesso ao ficheiros
test="$(date -d "$OPTARG" +%s &> /dev/null)"
if [[ $? -eq 0 ]]
then
    opd="$(date -d "$OPTARG" +%s)"
else
    echo "Opção -d "$OPTARG" não é uma data"
fi
echo "-d: $OPTARG"
;;
r ) #Sort Menor ao maior
echo "-r"
opr=1
;;
a ) #Sort Maior ao menor
echo "-a"
opa=1
;;
* ) #default
printf "Argumento Invalido!"
exit 1
;;
esac
done
shift $(( OPTIND - 1 ))
```

A imagem abaixo é referente a verificação de argumentos do nespace.sh aonde podemos ver que pode ser usado o argumento, -e, que acrescenta uma funcionalidade de indicação de uma lista de ficheiros considerados essenciais e que por isso se considera que não devem ser contabilizados ao determinar o espaço ocupado em disco.

```
;;
e ) #Lista de ficheiros a nao considerar
ope="$OPTARG"
echo "-e: $OPTARG"
```

```

### funcao para -l
func_l()
{
    ## Alocação de variáveis locais
    local Ndir="$(ls -l "$1" | grep -c ^d)" # N.º de subdiretórios

    local Nfile="$(ls "$1" -ltu --time-style=+%s | sort -k6 | grep ^*$4$ | awk -v adate="$3" '{if($6>adate) print;}'| grep -c ^-)" # N.º de ficheiros

    local dir_size # Tamanho do diretório

    # Se o diretório não tiver ficheiros, o n.º de bytes associado ao diretório é 0, se tiver, saca a soma dos bytes dos l maiores ficheiros
    if [[ "$Nfile" -eq "0" ]]; then
        dir_size="0"
    else
        # verificar se o $2 tem parametro numerico
        local l_select
        if [[ "$2" -eq "l" ]]; then
            l_select="$Nfile"
        else
            l_select="$2"
        fi
        dir_size="$(ls "$1" -ltu --time-style=+%s | sort -k6 | grep ^*$4$ | awk -v adate="$3" '{if($6>adate) print;}'| grep ^- | awk '{print '\$5'}' | sort -n | tail
        -"$l_select" | paste -sd+ | bc)"
    fi

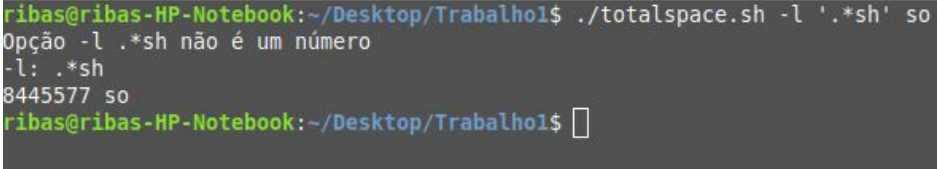
    local rel_dir="$1"

    size="$dir_size"
    echo "$dir_size" "$rel_dir"

    return 0
}

```

Nesta imagem podemos ver como foi feita a função para -l que serve para ver de quais os maiores ficheiros da diretoria que devem de ser considerados aonde fazemos a alocação das variáveis, ou seja o número de subdiretórios, o número de ficheiros e o tamanho do diretório. Vemos também que se o diretório não tiver ficheiros, o número de bytes associados ao diretório passa a ser 0, e se tiver conteúdo faz a soma dos bytes dos maiores ficheiros.



```

ribas@ribas-HP-Notebook:~/Desktop/Trabalho1$ ./totalspace.sh -l '*.sh' so
Opção -l *.sh não é um número
-l: *.sh
8445577 so
ribas@ribas-HP-Notebook:~/Desktop/Trabalho1$

```

Como podemos ver na imagem tivemos alguns problemas na correta realização do argumento onde este não consegue ver o subdiretórios que estão dentro do diretório “so”.

```
#####
## funcao para -L
func_L ()
{
    ## Alocação de variáveis locais
    local Ndir=$(ls -l "$1" | grep -c ^d) # N.º de subdiretórios

    local Nfile=$(ls "$1" -ltu --time-style=+%s | sort -k6 | tail -n +2 | grep ^"$4"$ | awk -v adate="$3" '{if($6>adate) print;}'| grep -c ^-)" # N.º de ficheiros

    # verificar se o $2 tem parametro numerico
    local l_select
    if [[ "$2" -eq "i" ]]; then
        l_select=$Nfile
    else
        l_select="$2"
    fi

    # Guardar os tamanhos dos ficheiros num array
    local rel_dir="$1"
    ls "$1" -ltu --time-style=+%s | sort -k6 | tail -n +2 | grep ^"$4"$ | grep ^- | awk -v adate="$3" '{if($6>adate) print;}'| awk -v var="$rel_dir" '{for(i=8;i<=NF;i+
+)$7=$7 OFS $i; print $5 " " var"/"$7}' | sort -k1n | tail -"$l_select"
}
```

Na imagem seguinte podemos ver como foi feita a função para -L que serve para ver de quais os maiores ficheiros de todas as diretorias que devem de ser considerados aonde fazemos a alocação das variáveis locais e vemos o número de subdiretórios onde guarda a informação sobre o tamanho dos ficheiros num array.

## Conclusão

Com este trabalho conseguimos desenvolver mais as nossas capacidades em bash, uma linguagem que não estamos muito habituados e não temos as mesmas capacidades que outras, mas no final sentimos que foi recompensador.

No início tivemos algumas dificuldades na maneira em que íamos manipular os dados como fazer. No momento o algoritmo não consegue procurar subdiretórios dentro do directório indicado, tentamos usar a recursividade sempre que encontrava um novo directório mas causava erro ou não conseguíamos somar o número de bytes.

Num trabalho futuro existem certos pontos que podiam ser mais aprofundados de maneira a recursividade funcione e mostre todos os subdiretórios e o desenvolvimento dos argumentos ser o mais correto.