universidade de aveiro

Ferramentas de descoberta do espaço ocupado em disco

Por: Vasco Ramos - nº mec 88931
Diogo Silva - nº mec 89348

Data de Preparação: Aveiro, 14 de Novembro de 2018

Cadeira: Sistemas Operativos

Corpo Docente: Professor Nuno Lau

Índice

Introdução

Em seguimento do plano curricular da disciplina Sistema Operativos, do curso de Engenharia Informática, da Universidade de Aveiro lecionada pelo Professor Doutor Nuno Lau, este relatório é o resultado da execução do primeiro trabalho da componente prática.

Intitulado de "Ferramentas de descoberta do espaço ocupado em disco", este projeto surge no seguimento de uma breve exposição à linguagem de programação BASH e tem como objetivo o melhor conhecimento e aprendizagem mais robusta e eficiente do uso da mesma.

Relativamente ao trabalho em si, o desafio era desenvolver scripts em bash que permitissem descobrir o espaço ocupado em disco por ficheiros com determinadas propriedades e que pudessem ser candidatos a ser apagados. A ferramenta a desenvolver deveria permitir vários modos de atuação, sendo um deles eliminar da contabilização do espaço ocupado por certos ficheiros que não serão contabilizados devido a serem considerados essenciais.

Para além disto, o trabalho permite também o uso da função getopts, que, até há data, ainda não tinha sido utilizada em qualquer outro programa em BASH realizado nas aulas.

Assim, o trabalho proposto encontra-se desenvolvido nos dois ficheiros criados por nós: **totalspace.sh** e **nespace.sh.**

Opções

Tanto a utilização do script **totalspace.sh** como a do **nespace.sh** requerem que o utilizador passe como argumento, pelo menos, um diretório. É também fornecido ao utilizador um conjunto de opções extra que controlarão que tipo de ficheiros devem ser considerados ou de que forma é que o output deve ser apresentado.

O conjunto de todas as opções permitidas é:

- **-n**: ficheiros com a expressão passada pelo utilizador
- **-l**: maiores ficheiros por diretório
- **-L:** maiores ficheiros no total
- **-d:** data de acesso
- **-r:** ordem inversa
- **-a:** ordem alfabética
- **-e:** exclui os ficheiros passados no ficheiro (apenas existe no nespace.sh)
- **-h:** mensagem de ajuda ao utilizador

É importante denotar que o **-l** e **-L** **não** podem ser utilizados em simultâneo, sendo apresentada uma mensagem de erro e terminado o script.

No caso em que ambas as opções de sorting, **-a** e **-r**, sejam especificadas, o output do script estará organizado alfabeticamente por ordem decrescente. Por outro lado, caso **nenhuma das duas seja especificada** o output estará organizado por ordem decrescente de tamanho.

A opção **-e** está apenas disponível no script nespace.sh.

A opção **-h** imprime como utilizar o script e termina a execução do script.

Comandos	Breve Explicação
./totalspace.sh sop **(1)** ./totalspace.sh -r sop **(2)** ./totalspace.sh -a sop **(3)**	**(1)** Ao executar o script sem qualquer opção, todos os ficheiros serão analisados e a ordem de impressão serã numérica (tamanho das pastas) decrescente. **(2)** Caso seja adicionada a opção "-r", a ordenação será feita tendo em conta na mesma os valores (a primeira coluna), ou seja, a ordenação será numérica, mas de forma crescente. **(3)** Caso seja adicionada a opção "-a", a ordenação será feita tendo em conta o nome dos diretórios (a segunda coluna), ou seja, a ordenação será alfabética de forma crescente.
./totalspace.sh -n ".*sh" sop **(1)**	**(1)** Caso seja adicionada a opção "-n", contabiliza-se apenas ficheiros que contenham a expressão *regex* passada pelo utilizador; neste exemplo a expressão regex é: ".*sh".
./totalspace.sh -l 2 sop **(1)**	**(1)** Caso seja adicionada a opção "-l", contabiliza-se apenas os *x* maiores ficheiros de cada diretoria, sendo *x* um número passado pelo utilizador.
./totalspace.sh -L 2 sop **(1)**	**(1)** Caso seja adicionada a opção "-L", contabiliza-se apenas os *x* maiores ficheiros de todas as diretorias, sendo *x* um número passado pelo utilizador, e imprime-os (junto com os seus tamanhos) como output.
./totalspace.sh -d "Sep 10 10:00" sop **(1)**	**(1)** Caso seja adicionada a opção "-d", contabiliza-se apenas os ficheiros que tenham sido acedidos até à data especificada pelo utilizador. Neste exemplo, vão ser considerados apenas ficheiros cujos o último acesso tenha sido feito até ao dia 10 de Setembro às 10:00.
./nespace.sh -e filelist sop **(1)**	**(1)** Caso seja adicionada a opção "-e", não se contabiliza os ficheiros que estejam incluídos no ficheiro especificado pelo utilizador.
./totalspace.sh -h **(1)** ./nespace.sh -h **(1)**	**(1)** Caso seja passada a opção "-h", vai ser impresso um menu de ajuda para o utilizador, que lhe dá alguma instruções de utilização da ferramenta.

Abordagem Base

getOpts

Primeiramente os nossos scripts fazem uso de um **getopts** para percorrer todas as opções passadas pelo utilizador.

```
while getopts ":l:d:L:n:arh" opt; do
  case $opt in
```

A variável opt vai ficar com o valor da opção encontrada e, sendo que caso encontre uma opção que não conste nas permitidas, será impressa a mensagem de **Usage** e o programa terminará.

Para cada uma das opções é declarada uma variável com o nome igual ao da opção chamada. Estas variáveis serão utilizadas como forma de verificação, durante o decorrer do programa, de que opções foram selecionadas. São também efetuadas operações de verificação de erros para garantir que a opção foi chamada em conformidade com o tipo de argumento que precisa. Por exemplo, na opção ***-l***, queremos garantir que o utilizador fornece o número de ficheiros (de maior tamanho) que quer considerar em cada diretoria, logo é favorável fazer uma verificação para garantir que é fornecido ao -l um número, e não uma letra (ou nada).

```
l)
    declare -a allArgSizes
    l=$OPTARG

    if ! [[ $l =~ ^[0-9]+$ ]] ; then
        echo "Error!"
        usage
        exit 1
    fi
    ;;
```

Por outro lado, é verificado caso o ***-l*** e o ***-L*** foram ambos ativados, sendo que o programa vai imprimir a mensagem de **Usage** e terminar se for o caso.

```
if [[ ! -z "$l" && ! -z "$L" ]]; then
    echo "Error! Only -l OR -L can be activated at a time"
    usage
    exit 1
fi
```

As opções de *sort*, -a e -r, foram utilizadas. Para estas opções, é alterada a variável **sortType** para que guarde o tipo de sort correspondente ("**-n"** para o caso da opção ***-r*** e "**-k 2"** para o caso da ***-a***), logo se ambas forem ativadas, a variável tomará o valor **"-rk 2"** (*sort* alfabeticamente por ordem inversa).

Main

Consideramos como "**Main**" a parte do código que efetivamente vai ler os argumentos correspondentes aos diretórios a percorrer, chamar a função findFiles, e imprimir no terminal os diretórios (ou dos ficheiros, no caso do ***-L***) e os seus tamanhos.

Começamos por percorrer cada um dos argumentos (Note que para evitar que as opções fossem lidas como argumentos, fizemos uso do comando shift **$((OPTIND-1))**) atribuindo-o à variável ***arg***. Verificamos então se ***$arg*** é um diretório (e se existe), sendo impressa uma mensagem de erro e a do **Usage**, caso não seja. Antes de mais, é também testado se temos permissão de leitura sobre o diretório passado em ***$arg***, e caso não tenhamos é imediatamente impresso que o seu tamanho é **NA** visto que será impossível determinar o seu tamanho.

Depois destas verificações, vai ser declarado um array associativo, ***directories***, onde recursivamente, serão guardadas, como *keys*, o ***$arg*** e o nome dos seus subdiretórios, e como *values*, os tamanhos correspondentes. São também inicializadas duas variáveis, **argSize** e **argError**, que serão utilizadas mais posteriormente para guardar o tamanho do diretório guardado em ***$arg*** e para ser mantido em conta se algum dos diretórios dentro do ***$arg*** é **NA** (logo o ***$arg*** também será **NA**).

Vai ser então chamada a função **findFiles** que determinará os tamanhos e colocará no ***directories*** as *keys*/*values* a serem impressas. Quando voltarmos da função vai ser

imediatamente verificado se o ***-l*** foi utilizado, visto que este utiliza uma rotina diferente da normal para determinar o tamanho total das diretorias (explicado à frente).

```
findFiles "$arg"

######################## l ######################
if [[  ! -z "$l" ]]; then
    getBiggestFiles "${allArgSizes[@]}" $argSize
    argSize=$thisSize
    subDirSearch $arg
    argSize=$(( $argSize + $thisDirSize ))
#################################################
fi
```

Quando já sabemos os tamanhos totais, vamos então verificar se a opção ***-L*** não foi ativada, visto que, pela maneira como esta funciona, antes de imprimir os maiores ficheiros, temos de percorrer todos os argumentos passados ao script, logo só queremos imprimir os tamanhos neste momento caso não estejamos a lidar com o ***-L*** (lembra-se que o ***-L x*** imprime os *x* maiores ficheiros de todas as diretorias (e suas subdiretorias) que passarmos como argumentos).

```
######################## NO L ###################
if [[ -z "$L" ]];then
        if [[ $argError == 1 ]];then
            directories+=([$arg]="NA")
        else
            directories+=([$arg]=${argSize})
        fi
        for i in "${!directories[@]}"
        do
            echo "${directories[$i]} ${i}"
        done
fi
#################################################
```

São então impressos os tamanhos da diretoria $arg e das suas subdiretorias

Note-se também que todo este processo é feito em pipeline com um **| sort $sortType** no final que ordenará a ordem dos prints pela ordem especificada.

```
329     ######################## L ######################
330
331     if [[ ! -z "$L" ]];then
332         for (( i=0; i<$L && i<${#bigFiles[@]}; i++ ))
333         do
334             echo ${bigFiles[$i]}
335         done
336
337     fi
```

findFiles

A função **findFiles** vai receber como argumento um diretório e irá percorrer, primeiro, todos os ficheiros do diretório e adicionar, num caso normal, o tamanho deste ao tamanho total do diretório. Ao percorrer cada ficheiro, vai também verificar se é possível determinar o tamanho do ficheiro e se o ficheiro em questão está em conformidade com todas as opções usadas. Temos, porém, dois casos em que a leitura dos ficheiros terá um comportamento diferente:

-l: Neste caso, em vez de ser adicionado o tamanho do ficheiro à variável **dirSize** (que guarda o tamanho do diretório atual), será colocado no array **allSizes** (ou **allArgSizes**, no caso se estivermos a percorrer os ficheiros do $arg).

```
39              ######################### -l #####
40          if [[  ! -z "$l" ]]; then   #Check if
41              if [[ -z "$currentDir" ]]; then #N
42                  allArgSizes+=($value)   #Add t
43                  continue
44              fi
45              allSizes+=($value)  #Add the file
46              ###################################
```

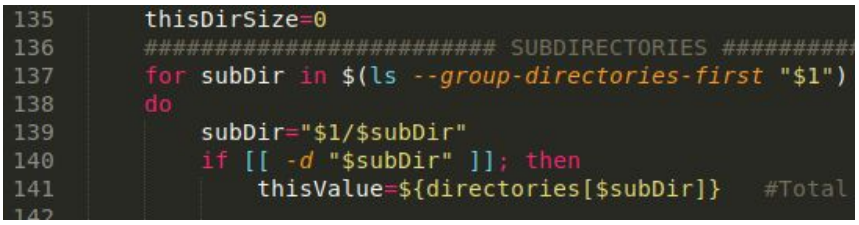
-L: Em vez de ser adicionado o tamanho do ficheiro à variável dirSize, será colocado o ficheiro no array **bigFiles** (inicializado quando a opção ***-L*** foi apanhada no **getOpts**) que guardará todos os ficheiros de todas as diretorias e os seus tamanhos, ordenadamente por ordem decrescente.

```
50  if [[ ! -z "$L" ]]; then
51      duplicate=0
52
53      for element in "${bigFiles[@]}"     #Check if we already have
54      do
55          if [[ $element == "$value $file" ]];then
56              duplicate=1
57              continue
58          fi
59      done
60
61      if [[ $duplicate == 1 ]]; then
62          continue
63      fi
64
65      bigFiles+=("$value $file")
66      bigFiles=( $( printf "%s\n" "${bigFiles[@]}" | sort -n -r ) )
67      continue
68  fi
69  ######################################################
```

Depois de termos percorrido todos os ficheiros, vamos percorrer todos os subdiretórios e chamar recursivamente a função **findFiles** para determinar os tamanhos desses. Fica assim garantido que conhecemos os tamanhos de todos os subdiretórios, logo, depois disto vamos chamar a função **subDirSearch** para somar ao tamanho atual do diretório os tamanhos dos seus subdiretórios. É importante referir que vamos guardar três variáveis locais a esta iteração da função **findFiles**, o ***currentDir*** que guarda o path do diretório atual, o ***dirSize*** que guarda o tamanho do diretório atual e o ***error***, que é inicializado a 0 e mudará para 1 caso ocorra algum erro durante a leitura dos subdiretórios (ou se o próprio diretório não é *readable*).

subDirSearch

Esta função vai receber como argumento um diretório e vai ser usada para percorrer todos os subdiretórios deste e guardar os seus tamanhos numa variável chamada ***thisDirSize***, que depois será utilizada para somar ao valor das variáveis locais ***dirSize*** declaradas na função findFiles. O tamanho dos subdiretórios é conseguido procurando, iterativamente, no array associativo, ***directories***, o *value* correspondente à *key* igual ao *path* do subdiretório atual.



```
135     thisDirSize=0
136     ######################## SUBDIRECTORIES #########
137     for subDir in $(ls --group-directories-first "$1")
138     do
139         subDir="$1/$subDir"
140         if [[ -d "$subDir" ]]; then
141             thisValue=${directories[$subDir]}   #Total
```

getBiggestFiles

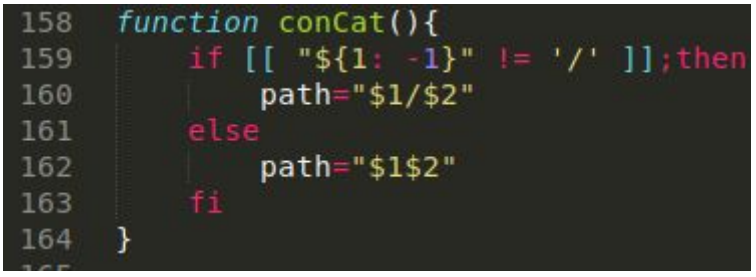
A **getBiggestFiles** é chamada para que possamos determinar os ***x*** maiores elementos de um array. É utilizada quando a opção ***-l*** está ativa para que possamos ter em consideração no tamanho dos diretórios apenas os ***x*** maiores ficheiros (sendo ***x*** o *OPTARG* passado à opção ***-l***).

```
166 function getBiggestFiles(){
167     thisAllSizes=( $( printf "%s\n" "$@" | sort -n -r) )
168     thisSize=0
169
170     for (( i=0; i<"$l" && i<"${#thisAllSizes[@]}"; i++ ))
171     do
172         thisSize=$(( $thisSize + ${thisAllSizes[$i]} ))
173     done
174 }
```

conCat

A função **conCat** recebe duas variáveis e verifica se o último caracter da primeira é ***"/"***.

Caso não seja, concatena as duas variáveis na forma stringA/stringB, e caso seja, concatena na forma stringAstringB. De ambas as formas, o resultado da concatenação é guardado na variável ***path***.



Problemas Encontrados

Durante a elaboração dos scripts, depará-mo-nos com vários problemas, notavelmente, os seguintes foram os mais desafiantes:

- Salvaguarda dos valores dos diretórios/tamanhos correspondentes.

- Como percorrer primeiro os ficheiros e só depois os subdiretórios.

- Como adicionar ao tamanho de um diretório os tamanhos dos seus subdiretórios.

- Como garantir que o tamanho do diretório atual não é perdido ao iniciarmos a recursividade do findFiles sobre os seus subdiretórios.

- Tratamento de erros e atribuição do valor NA a diretórios cujos ficheiros não consigamos determinar o tamanho ou que não sejam acessíveis.

Fases de Abordagem

No decorrer da abordagem ao problema que nos foi proposto, adotámos algumas ideias algo distintas.

Inicialmente, aquando da realização da função de cálculo do tamanho de cada diretório (sem qualquer opção) a abordagem usada foi uma stack, em que cada entrada correspondia ao valor de uma diretoria, no entanto, de forma a conseguirmos tornar a solução mais modular e de fácil adaptação às várias opções acabámos por alterar esta ideia, e portanto partimos para a utilização de um array associativo para guardar como keys os path dos diretórios, e como values os seus tamanhos.

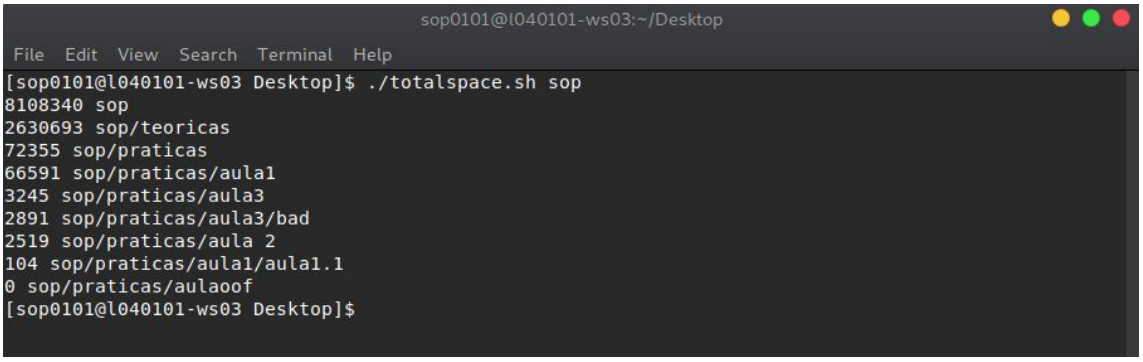
Depois disto tivemos que arranjar maneira de adicionar os tamanhos dos subdiretórios ao diretório atual. Para tal, a melhor forma que encontrámos foi chamar a função findFiles recursivamente para os subdiretórios após termos já determinado o tamanho do diretório atual. Desta forma garantimos que temos a soma do tamanho dos ficheiros do diretório, logo os subdiretorios terão os tamanhos corretos. Isto porém levantou o problema de "Como é que iríamos percorrer primeiro todos os ficheiros de um diretório e só depois passar aos seus subdiretórios".

Inicialmente utilizava-mos um único ciclo for que, com o uso de um ls -l, nos permitia filtrar os diretórios e depois os ficheiros como o uso de um awk e vários grep's. O problema com esta implementação era o facto de a condição do for ficar enorme, e no caso dos links simbólicos, o nome apanhado pelo for estaria no formato "abc -> abc", o que criaria problemas de "file not found" aquando do stat. Então, após mais pesquisa sobre as várias opções do ls, mudamos para um ls -p que coloca um "/" no final de cada os subdiretórios e depois com um uso do grep, filtramos para ignorar elementos que incluíssem o "/" (o que não criaria problemas visto que isto é um caracter ilegal para o nome dos ficheiros).

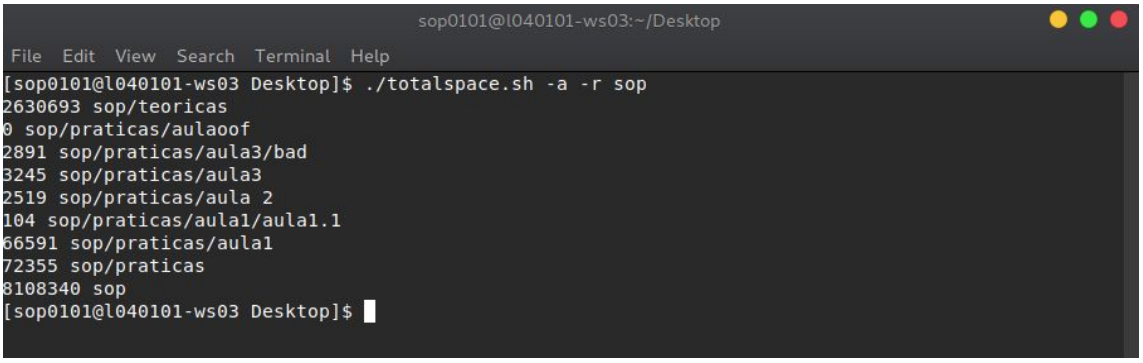
A última fase de abordagem consistiu em remover algum código repetido da função findFiles e colocá-lo em funções distintas, como, por exemplo, a rutina que percorria os subdiretórios de um diretório para adicionar o tamanho destes ao tamanho do diretório (sendo que esta rutina foi transformada na função **searchSubDirs**).

Testes da Execução do Programa

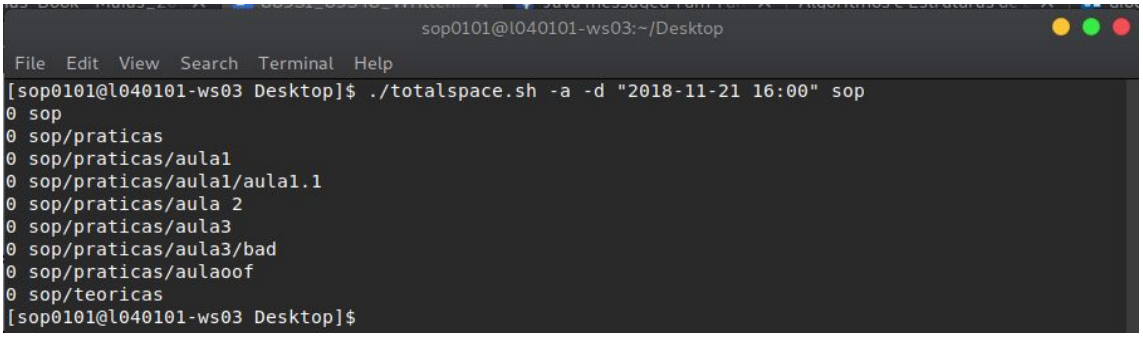
./totalspace.sh sop



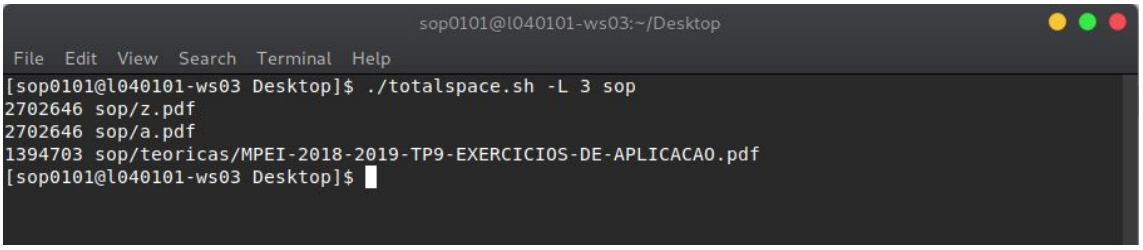
./totalspace.sh -a -r sop



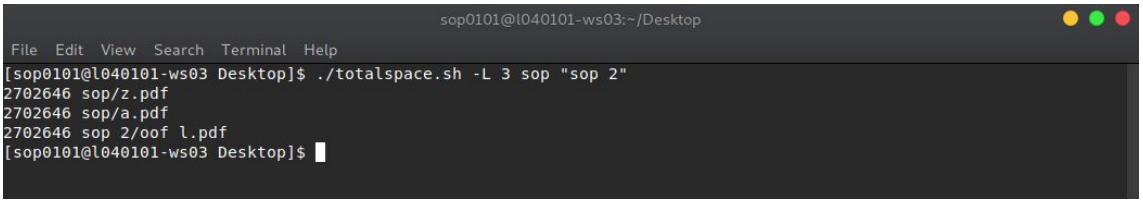
./totalspace.sh -a -d “2018-11-21 16:00” sop



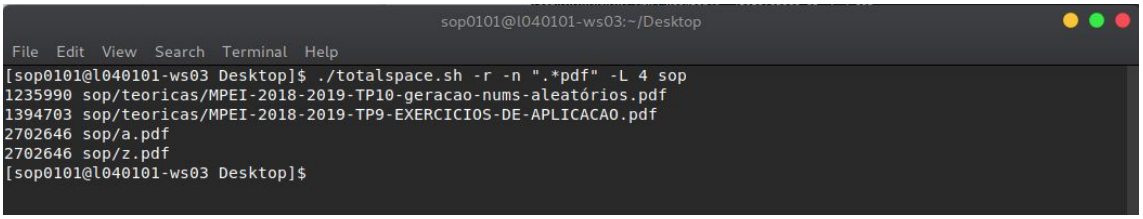
./totalspace.sh -L 3 sop



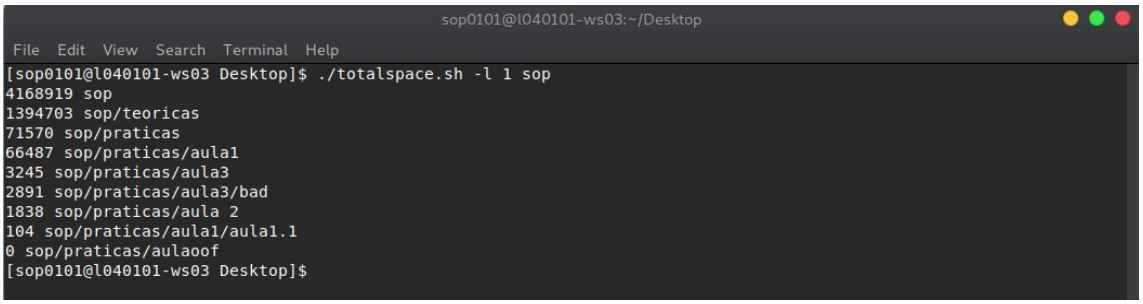
./totalspace.sh -L 3 sop “sop 2”



./totalspace.sh -r -n “.*pdf” -L 4 sop

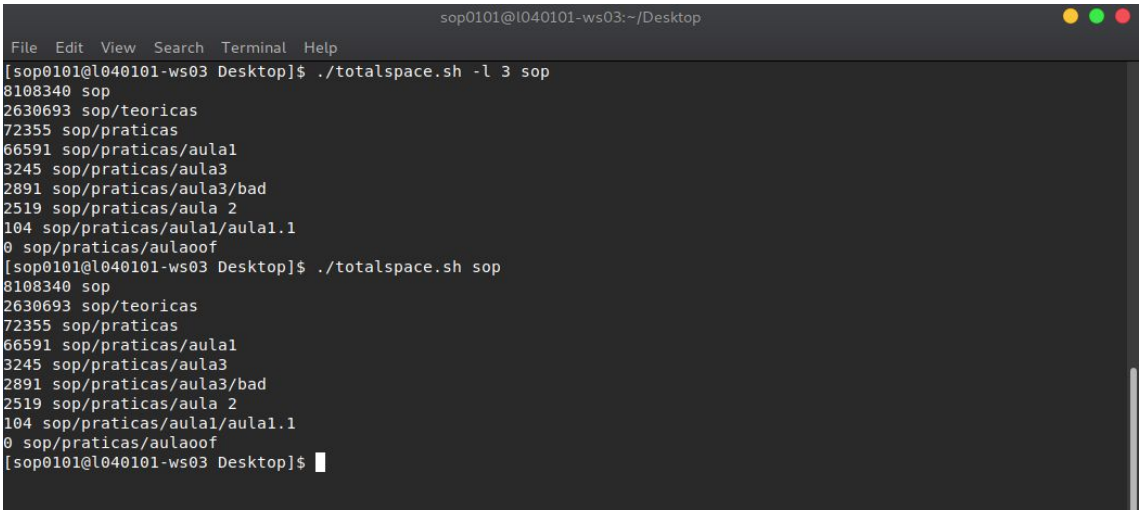


./totalspace.sh -l 1 sop

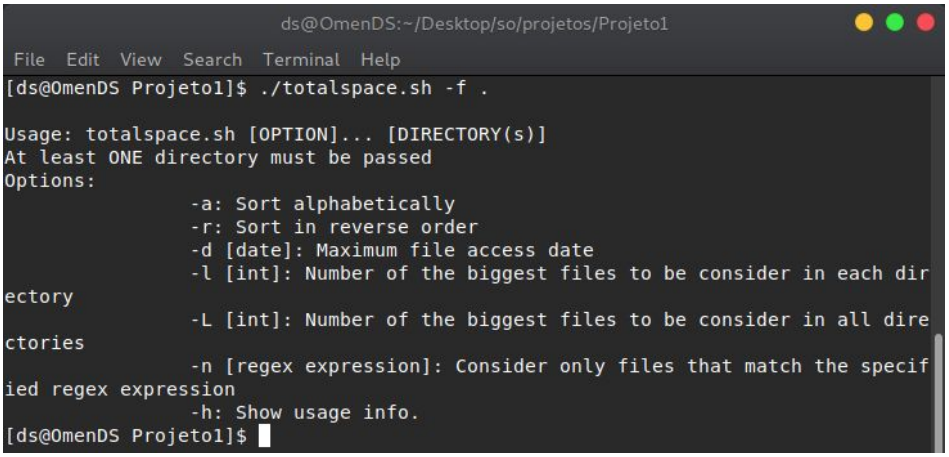


./totalspace.sh -n ".*pdf" sop "sop 2"

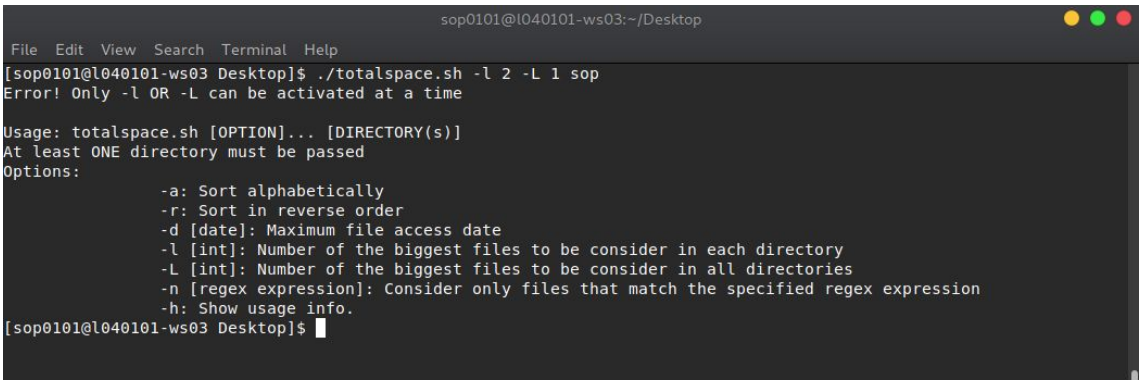
./totalspace.sh -l 3 sop



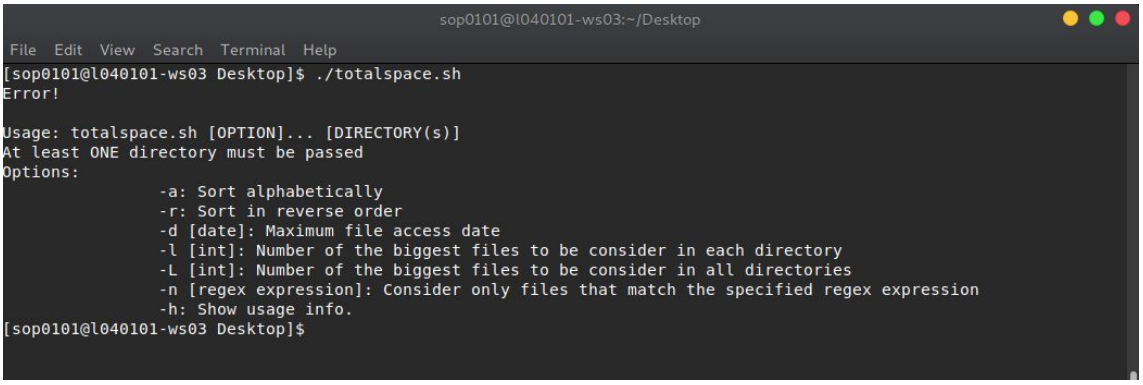
./totalspace.sh -f .



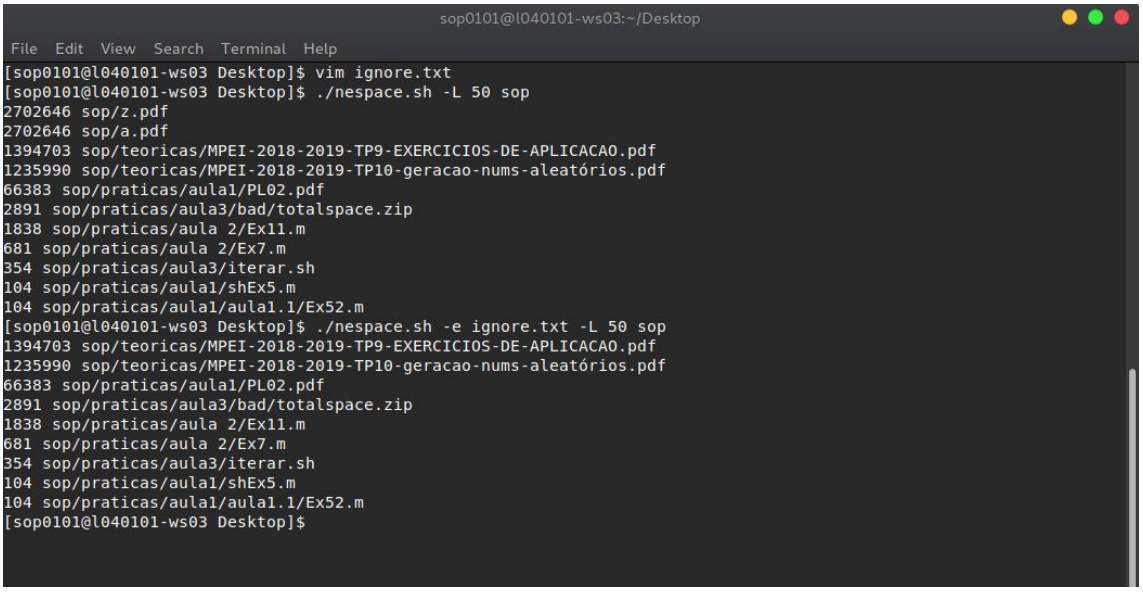
./totalspace.sh -l 2 -L 1 sop



./totalspace.sh



./nespace.sh -e ignore.txt -L 50 sop (& ./nespace.sh -L 50 sop)



Considerações Finais

Em retrospetiva, consideramos que todos os objetivos deste trabalho foram cumpridos. Pensamos que conseguimos implementar todas as funcionalidades exigidas, sendo que, segundo os nossos testes, todas as opções passadas pelo utilizador estão a ter o resultado pretendido.

Em suma, concluímos que, com este trabalho, melhorámos os nossos conhecimentos sobre bash e desenvolvemos a nossa capacidade de pesquisa e autonomia.

Bibliografia

- http://wiki.bash-hackers.org/
- http://www.tldp.org/guides.html
- https://stackoverflow.com/
- https://unix.stackexchange.com/
- https://www.thegeekstuff.com/