deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

Projeto 1 de Sistemas Operativos

Gestão de armazenamento: Monitorização do espaço ocupado

Trabalho realizado por:

Ricardo Martins, nº112876

Rodrigo Jesus, nº113526

Ano letivo 2023/24

ÍNDICE

Introdução

O objetivo do trabalho é o desenvolvimento de scripts em bash que permitam monitorizar o espaço ocupado em disco (em bytes) por todos os ficheiros selecionados em todas as diretorias passadas como parâmetro e nas suas descendentes, bem como a sua variação ao longo do tempo facilitando assim a gestão do armazenamento.
Para tal foram criados dois ficheiros principais: spacecheck.sh para realizar uma análise de diretórios e gerar um relatório do espaço ocupado, e spacerate.sh para comparar dois desses relatórios e identificar possíveis alterações no espaço ocupado pelos diretórios.

Metodologia

A metodologia adotada para abordar esse projeto envolveu uma análise cuidadosa dos requisitos, implementação incremental, testes rigorosos e a consideração de boas práticas de programação em Shell.

Funcionalidades

spacecheck.sh:

- Calcula e exibe o espaço ocupado por diretórios, filtrando por nome, data e tamanho.
- Suporta opções de ordenação, por tamanhos ou por nomes, e limitação do número de linhas exibidas.
- Lida com erros de forma informativa.

spacerate.sh:

- Compara dois relatórios gerados pelo spacecheck.sh.
- Identifica novos diretórios, remoções e alterações no espaço ocupado.
- Aceita opções de ordenação, por tamanho ou por nomes, para saída personalizada de output.
- Lida com erros de forma informativa.

Erros e avisos

Para além de todas as funcionalidades inerentes os scripts, ambos foram também projetados para lidar de forma correta com uma vasta gama de erros e exceções que possam ocorrer na execução do programa. Nomeadamente:

1. A introdução de um ficheiro não existente como argumento
2. O ficheiro fornecido como argumento não ser do tipo correto
3. O ficheiro fornecido não tem permissões de acesso, ou seja está invisível
4. Inserir comandos diferentes dos disponíveis.
5. Inserir o mesmo comando várias vezes
6. Inserir valores inválidos para os diferentes comandos, ex: inserir uma data invalida depois do -d; inserir um número menor do que 0 depois do -s
7. Não inserir os argumentos de entrada na ordem correta, devem colocar primeiro todas as funcionalidades desejadas (“-r” ; “-a” ; “-l”) e só depois as diretorias.
8. Para o spacecheck.sh, sempre que, por alguma razão (falta de permissões, por exemplo), não seja possível aceder a uma diretoria o espaço ocupado pela diretoria deve ser NA.

Exemplo de uma mensagem de erro que é lançada quando o ficheiro fornecido no input não é um diretório:

```
Test Case 7: Inaccessible directory
Error: The argument: unreadable_folder is not a directory

Usage: ./spacecheck.sh [-n <regex>] [-d <date>] [-s <size>] [-r] [-a] [-l <limit>] [<directory1> <directory2> ...]
  -n <regex>   Filter files by name using a regular expression
  -d <date>    Filter files by modification date (e.g., 'Sep 10 10:00')
  -s <size>    Filter files by minimum size in bytes
  -r           Sort output in reverse order
  -a           Sort output by name (default is by size)
  -l <limit>   Limit the number of lines in the output
  <directory>  Directories to check
```

Explicação do código – Spacecheck.sh

```
# Initialize variables for the options
regex=".*"
limit=999999
min_size=0
max_date=$(date "+%b %d %H:%M")
sort_flag="-nr" # default value, sorts by sizes in descending orde
filters=""
```

As variáveis são inicializadas com valores padrão, como uma expressão regular para filtrar por nome (regex), limites para tamanho mínimo de arquivo (min_size) e data máxima de modificação (max_date). A flag de ordenação (sort_flag) é configurada para ordenar por tamanho em ordem decrescente por padrão.

```
# Function to calculate directory size and print the result
calculate_directory_size() {
  local directory="$1"
  local total_size=0

  # Calculate the total size of files in the directory
  total_size=$(find "$directory" -type f -not -newermt "$max_date" -size +"$min_size"c -regex "$regex" -exec stat -c %s {} \; 2>/dev/null | awk '{s+=$1} END {print s}')

  # If the directory is not readable set the total size to NA
  if [ ! -r "$directory" ]; then
    total_size="NA"
  fi

  # If the total size couldn't be determined, set it to 0
  if [ -z "$total_size" ]; then
    total_size=0
  fi

  printf "%-10s %-10s\n" "$total_size" "$directory"
}
```

Função de Cálculo de Tamanho do Diretório (calculate_directory_size**):**
Essa função recebe um diretório como argumento e calcula o tamanho total dos arquivos nesse diretório, filtrando por nome, data e tamanho conforme especificado nas opções. O resultado é exibido no formato "tamanho nome_do_diretório".

```
while getopts ":n:d:s:l:ra" opt; do
```

O bloco while getopts processa as opções da linha de comando usando o comando getopts. As opções disponíveis incluem filtros por nome, data e tamanho, opções de ordenação e limitação do número de linhas na saída.
O bloco de getopts possui ainda verificação de erros quanto ao uso das diferentes opções.

```
for dir in "$@"; do
  if [ ! -d "$dir" ]; then
    echo "Error: The argument: $dir is not a directory"
    printf "\n"
    usage
  fi
done
```

O primeiro loop for dir in "$@" percorre todos diretórios fornecidos na linha de comando à procura de algum que não seja diretório, caso encontre algum lança a função usage que termina o programa e imprime como devem ser passados os argumentos de entrada no terminal.

```
# Loop through directories and their subdirectories
for dir in "$@"; do
  output_file="${dir}_$(date "+%Y%m%d")"

  # Print the header line to the file and the terminal
  printf "%s %s $(date "+%Y%m%d") %s %s\n" "SIZE" "NAME" "$filters" "$dir" | tee "$output_file"

  # Calculate the sizes of subdirectories and print the result
  while IFS= read -r -d $'\0' subdirectory; do
      calculate_directory_size "$subdirectory"
  done < <(find "$dir" -type d -print0 2>/dev/null) | sort $sort_flag | head -n $limit | tee -a "$output_file"
  printf "\n"

done
```

O segundo loop for dir in "$@" percorre os diretórios fornecidos na linha de comando. Para cada diretório, é criado um arquivo de saída com o formato "nome do diretório_data". O cabeçalho é impresso no arquivo e na tela, seguido pelos cálculos de tamanho para os sub diretórios.

Durante o processamento, são realizadas verificações para garantir que os diretórios existam. Os resultados são exibidos no formato especificado, ordenados e limitados conforme necessário.

Exemplo de um output do programa para a linha de comandos:

./spacecheck.sh -n ".*txt" -r Folder1

```
SIZE NAME 20231113 -n .*txt -r  Folder1
0          Folder1/Folder2/Folder1
0          Folder1/Folder2/Folder5/Folder1
102        Folder1/Folder2/Folder5/Folder2
1023       Folder1/Folder2/Folder5/Folder4/Folder4
1918       Folder1/Folder2/Folder2/Folder1
1972       Folder1/Folder2/Folder5/Folder3/Folder1
2137       Folder1/Folder1/Folder2
2281       Folder1/Folder2/Folder2/Folder2/Folder1
2363       Folder1/Folder2/Folder5/Folder4/Folder2
2377       Folder1/Folder1/Folder1/Folder2
2408       Folder1/Folder2/Folder5/Folder4/Folder3
3151       Folder1/Folder2/Folder5/Folder3
3382       Folder1/Folder2/Folder5/Folder4/Folder1
3462       Folder1/Folder1/Folder1/Folder1
4360       Folder1/Folder2/Folder2/Folder2/Folder2
4889       Folder1/Folder1/Folder1/Folder3
9405       Folder1/Folder2/Folder5/Folder4
9446       Folder1/Folder2/Folder2/Folder2
12715      Folder1/Folder2/Folder2
13243      Folder1/Folder2/Folder5
14463      Folder1/Folder1/Folder1
19704      Folder1/Folder1
27543      Folder1/Folder2
54842      Folder1
```

Explicação do código – spacerate.sh

Primeiras considerações:

- Os últimos 2 argumentos têm de ser os ficheiros output do spacecheck;
- O primeiro ficheiro é considerado o mais recente;
- Os ficheiros não podem ter extensão;
- Se um dos ficheiros tiver as flags "-a" e/ou "-r", a flag é sempre assumida como um "modificador" para a ordenação (caso não seja inserida nenhuma flag como argumento pelo utilizador).

```
if [ $# -eq 2 ]; then # If the number of arguments is 2, then proceed to call the verification
function and test $1 and $2 (the last 2 arguments have to be the files with the spacecheck
output)
  output=$(verification "$1" "$2")
  f1=$1
  f2=$2
elif [[ (( $1 == "-r" || $1 == "-a" )) && $# -gt 2 && $# -lt 5 ]]; then # If the number of
arguments is 3 and $1 is a "sort expression", then proceed to call the verification function and
test $2 and $3
  mod=$1
  if [ $# -eq 3 ]; then
    output=$(verification "$2" "$3")
    f1=$2
    f2=$3
  elif [[ $# -eq 4 && (( $2 == "-r" || $2 == "-a" )) ]]; then # If the number of arguments is 3
and $1 and $2 are a "sort expression", then proceed to call the verification function and test
$3 and $4
    output=$(verification "$3" "$4")
    f1=$3
    f2=$4
    mod2=$2S
  else
    echo "Error: Invalid type of arguments"
    exit 1
  fi
else
  echo "Error: Invalid number of arguments"
  exit 1
fi
```

A primeira coisa que o código spacerate.sh faz é verificar se o número de argumentos inserido é correto, isto é, existe um número mínimo de argumentos, que é 2 (os 2 ficheiros que são um output do spacecheck) e um número máximo que consideramos ser 4, os 2 ficheiros mais 2 flags, que no caso do spacerate, são as flags "-r" e "-a".
Os erros causados ao utilizar o spacerate estão espalhados ao longo do código para evitar operações redundantes e poupar o tempo do utilizador.

```
verification(){
  if [[ ! (( -e $1 && -e $2 )) ]]; then
    echo "Error: Non existent file(s)"
  else
    IFS="." read -r temp fExt1 <<< "$1"  # Get the extension of the argument
    IFS="." read -r temp fExt2 <<< "$2"
    fType1=$(file -b $1)
    fType2=$(file -b $2)
    if [[ ! (( $fType1 == *"text"* && $fType2 == *"text"*  && $fExt1 == "" && $fExt2 ==
"" )) ]]; then # Files type verification (plain text, doesn't include files like random.sh)
      echo "Error: File(s) in the wrong type"
    else
      IFS="/" read -ra arg1 <<< "$1" # To function even when the script is called from a
different "path" (example: SO/Projeto1/spacerate.sh instead of calling ./spacerate.sh)
      IFS="/" read -ra arg2 <<< "$2"
      i1=$((${#arg1[@]} - 1))
      i2=$((${#arg2[@]} - 1))
      IFS="_" read -r dir1 date1 <<< "${arg1[i1]}" # Get the directory name in the name of the
file
      IFS="_" read -r dir2 date2 <<< "${arg2[i2]}"
      if [[ $dir1 != $dir2 ]]; then
        echo "Error: The directories are different"
      else
        echo "$dir1 $date1 $date2" # Output the directory and both dates
      fi
    fi
  fi
}
```

A função “verification()” verifica os argumentos em si, o tipo deles, se os ficheiros existem, se o diretório é o mesmo e no fim devolve o diretório bem como ambas as datas, para serem usadas depois noutras verificações.

```
declare -A list1 list2 fArray newList removedList flags
fLine1=() # Holds the first line of the file
fLine2=()
args1=()
args2=()
flags["-r"]=0 # Counts how many times does a flag appear in both files
flags["-a"]=0
flags["-l"]=0
flags["-s"]=0
flags["-n"]=0
flags["-d"]=0
n=0 # Helps to save the conditions atatched to the flags (like "-s 123")
i=0 # Helps to separate the first and the rest of the lines
```

As variáveis são inicializadas de forma padrão, e foram usadas listas e arrays para ordenar, filtrar e verificar os ficheiros.

```
while IFS=$'\t' read -r line; do # Reads both files at the same time line by line
  if [ $i != 0 ]; then # Checks if it isn't the first line
    read -r p1 p2 <<< "$line" # Splits a line in 2, $p1 is the size and $p2 the directory name
    if [[ $p1 != "" && $p2 != "" ]]; then
      list1["$p2"]=$p1
    fi

  else # If read line is the first
    for s in $line; do
      fLine1+=("$s") # Save the contents of it in a list
      if [[ $s == "-r" || $s == "-a" || $s == "-l" || $s == "-s" || $s == "-n" || $s == "-d" ]];
then
        flags["$s"]+=1 # Adds 1 to the correspondent flag counter
        n=$((n+1))
      else
        args1[$n]+="$s"
      fi
    done

    lastIndex1=$((${#fLine1[@]} - 1)) # Gets the index of the last item of the first line
    dir1=${fLine1[$lastIndex1]} # Gets the directory name in the first line

    if [[ ${fLine1[0]} != "SIZE" || ${fLine1[1]} != "NAME" || ${fLine1[2]} != "$date1" ||
"$dir" != "$dir1" ]]; then # Check if the first line corresponds to a spacecheck output, if the
correspondent dates and directory name are the same as the ones in the name of the files
      echo "Error: File(s) is(are) not a spacecheck.sh output"
      exit 1
    fi

  fi
  i=$((i+1)) # Line counting
done < <(paste "$f1")
```

Usamos um while como o loop que lê o conteúdo dos ficheiros e os guarda em arrays (cada diretório tem o respetivo tamanho associado). Existe uma verificação dentro do loop que serve para verificar se o ficheiro é um output do spacecheck (é feita através da primeira linha, nomeadamente pela existência das strings “SIZE” e “NAME” e da correspondência entre o nome dos diretórios e as datas descritas na primeira linha com as do nome do ficheiro.

```
# Checks and adds to an array equal elements of both lists
for dir in "${!list1[@]}" "${!list2[@]}"; do
  if [[ -n "${list1[$dir]}" && -n "${list2[$dir]}" ]]; then
    dif=$((list1["$dir"] - list2["$dir"]))
    fArray["$dir"]=$dif

  # NEW directories
  elif [ -n "${list1[$dir]}" ]; then
    dif=$((list1["$dir"]))
    fArray["$dir"]=$dif
    newList["$dir"]="1" # Adds to a list the new directories

  # REMOVED directories
  elif [ -n "${list2[$dir]}" ]; then
    dif=$((0 - list2["$dir"]))
    fArray["$dir"]=$dif
    removedList["$dir"]="1" # Adds to a list the removed directories
  fi
done
```

Esta parte verifica se um diretório existe em ambos os ficheiros e faz a diferença entre tamanhos, se existir apenas no primeiro, então o ficheiro é novo (“NEW”), se existir apenas no segundo quer dizer que foi removido (“REMOVED”), adicionando a uma

lista os diretórios novos e a outra os removidos e os respetivos tamanhos (como são únicos, não existe diferença).

Exemplo de um output do programa para a linha de comandos:

./spacerate.sh Folder1_20231113 Folder1_20231112

```
SIZE NAME
-85243 Folder1/Folder2/Folder5 REMOVED
-81613 Folder1
-81613 Folder1/Folder2
-60264 Folder1/Folder2/Folder5/Folder4 REMOVED
-14677 Folder1/Folder2/Folder5/Folder4/Folder1 REMOVED
-14665 Folder1/Folder2/Folder5/Folder4/Folder3 REMOVED
-14384 Folder1/Folder2/Folder5/Folder4/Folder2 REMOVED
-14320 Folder1/Folder2/Folder5/Folder4/Folder4 REMOVED
-13981 Folder1/Folder2/Folder5/Folder3 REMOVED
-9791 Folder1/Folder2/Folder5/Folder3/Folder1 REMOVED
-2265 Folder1/Folder2/Folder5/Folder1 REMOVED
-610 Folder1/Folder2/Folder5/Folder2 REMOVED
0 Folder1/Folder1
0 Folder1/Folder1/Folder1
0 Folder1/Folder1/Folder1/Folder1
0 Folder1/Folder1/Folder1/Folder2
0 Folder1/Folder1/Folder1/Folder3
0 Folder1/Folder1/Folder2
0 Folder1/Folder2/Folder1
0 Folder1/Folder2/Folder2/Folder1
0 Folder1/Folder2/Folder2/Folder2
0 Folder1/Folder2/Folder2/Folder2/Folder1
0 Folder1/Folder2/Folder2/Folder2/Folder2
3630 Folder1/Folder2/Novo NEW
89216 Folder1/Folder2/Folder2 NEW
```

Neste caso, o output está por ordem crescente porque pelo menos num dos ficheiros está a flag “-r”.

Ficheiros testCases / testCases no terminal

Para testar todas estas funcionalidades foram criados 3 ficheiros scripts adicionais:
O primeiro o **generateFolders.sh** para criar um sistema aleatório de ficheiros e diretorias para servirem de argumento de teste para a função spacecheck.sh
Em segundo lugar criamos o script testCases.sh com todos os casos de teste que achamos necessários para demonstrar todas as capacidades do script spacecheck
Por fim criamos o script testCasesSR.sh que também possui todos os casos de teste que achamos necessários para demonstrar as capacidades do script spacerate.
Para testar o spacecheck utilizamos o script do generateFolders.sh no terminal

- **./generateFolders.sh** e de seguida usamos **./testCases.sh**

Para testar o spacerate.sh modificamos o script testCasesSR.sh para ter os nomes dos outputs fornecidos pelo script do spacecheck.sh e de seguida escrevendo **./testCasesSR.sh** no terminal.

Dificuldades encontradas

Na criação da função calculate_directory_size do ficheiro spacecheck não fomos capazes de encontrar uma forma mais eficiente para fazer o cálculo do tamanho dos diretórios e subsequente sub diretórios e ao mesmo tempo aplicar os filtros e ordenação necessários aos mesmos.
Já no ficheiro spacerate fomos capazes de implementar todas as funcionalidades que este devia possuir apesar de termos encontrado algumas dificuldades, nomeadamente para outputs do spacecheck em que ambos tinham as flags “-d”, “-s” ou “-n”.

Conclusão

Para concluir o projeto demonstrou a aplicação prática de Shell Script em cenários de administração de sistemas e monitorização de diretórios, tendo a resolução deste projeto sido um sucesso visto que conseguimos implementar todas as funcionalidades necessárias à correta execução do código e acabamos por adquirir uma vasta gama de conhecimentos sobre Bash, bem como a capacidade de criar soluções robustas e eficientes nesta linguagem.