Relatório SO Projeto 1

Ferramenta de criação/atualização de cópias de segurança em bash

Rafael Fernandes - 118956 **Rodrigo Santos - 119198**

2024/2025

Prof. José Nuno Panelas Nunes Lau

Índice

1 Introdução **3**
- 1.1 Backup Files (backup_files.sh) . . . 3
- 1.2 Backup (backup.sh) . . . 3
- 1.3 Backup Summary (backup_summary.sh) . . . 3
- 1.4 Backup Check (backup_check.sh) . . . 3

2 Backup Files **4**
- 2.1 Implementação . . . 4
- 2.2 Testes . . . 6

3 Backup **7**
- 3.1 Implementação . . . 7
- 3.2 Testes . . . 10

4 Backup Summary **12**
- 4.1 Implementação . . . 12
- 4.2 Testes . . . 15

5 Backup Check **16**
- 5.1 Implementação . . . 16
- 5.2 Testes . . . 17

6 Conclusão **18**

7 Bibliografia **19**

1 Introdução

O objetivo deste projeto foi o desenvolvimento de scripts em shell / bash (backup_files.sh, backup.sh, backup_summary.sh, backup_check.sh), que permitam realizar um backup de um diretório específico.

Estas ferramentas permitem fazer um backup de todos os ficheiros e subdiretórios do diretório principal para um diretório backup escolhido pelo utilizador.

Este relatório documenta as etapas de desenvolvimento, as decisões de adotadas, tal como as soluções implementadas para superar adversidades encontradas durante a implementação.

1.1 Backup Files (backup_files.sh)

Este script, **backup_files.sh** foi o primeiro script que desenvolvemos. Este tem como objetivo realizar o backup apenas dos ficheiros no diretório escolhido. Para além desta funcionalidade, temos a opção de *checking* **"-c"** que, quando dada como argumento, exibe os comandos que seriam executados para realizar o backup, sem realmente os executar.

1.2 Backup (backup.sh)

Melhorando o script anterior, criámos o script **backup.sh** com todas as funcionalidades do **backup_files.sh**, adicionando a recursividade que nos permite fazer backup tanto a ficheiros como a diretórios e aos ficheiros dentro dos mesmos. Neste ficheiro também implementámos 2 opções novas **"-b [*file*]"**, que ignora os ficheiros cujo nome esteja no ficheiro de texto dado como argumento e **"-r [*regex*]"** que copia apenas os ficheiros cujos nomes correspondam à expressão regular (*regex*).

1.3 Backup Summary (backup_summary.sh)

Expandido as funcionalidades de **backup.sh**, esta iteração teve várias adições significantes, entre as quais a remodelação completa da forma com o programa é executado, a detenção de prováveis casos problemáticos e a implementação de contadores, para mostrar informações relevantes ao utilizador, o que auxilia a verificação da integridade do processo.

1.4 Backup Check (backup_check.sh)

Este foi o último script que desenvolvemos. O script percorre o diretório e todos os subdiretórios de uma pasta destino e verifica se são iguais à pasta inicial. Usando o método ***md5sum***, o script verifica se os ficheiros com o mesmo nome são iguais, o que acaba por ser muito útil para verificar se o diretório *backup* está atualizada em relação ao diretório *source*.

2 Backup Files

2.1 Implementação

A primeira parte da nossa implementação apenas tem como objetivo verificar se o utilizador está a chamar corretamente o programa, mostrando a forma correta de se usar e acabando a execução do programa caso tenha mais que 3 ou menos que 2 argumentos.

```
1  #!/bin/bash
2
3  # Verifica se o número de argumentos é válido
4  if [[ $# -lt 2 || $# -gt 3 ]]; then
5    echo "Uso: $0 [-c] <src> <backup_dst>"
6    exit 1
7  fi
8
```

Após esta verificação o programa inicia a variável **CHECK_MODE** e atribui-lhe o valor "-c", caso o argumento [-c] tenha sido fornecido, removendo-o da lista de argumentos e colocando os diretórios de trabalho e backup nos argumentos 1 e 2 respetivamente.

```
9  # Modo de checking (por default copia os ficheiros)
10 CHECK_MODE=""
11
12 # Verifica se a opção -c foi fornecida
13 if [[ "$1" == "-c" ]]; then
14   CHECK_MODE="-c"
15   # Remove o primeiro argumento e deixa os outros no $1 e $2
16   shift
17 fi
18
19   # Atribui valores aos argumentos
20   SRC_DIR="$1"
21   BACKUP_DIR="$2"
22
```

Em adição à primeira verificação, temos as verificações para os diretórios, que acabam a execução do programa caso algum do diretórios tenha o nome vazio, ou caso o diretório de trabalho não exista.

```
23 # Verifica se o diretório de origem existe
24 if [[ ! -d "$SRC_DIR" ]]; then
25   echo "O diretório de origem '$SRC_DIR' não existe."
26   exit 1
27 fi
28
29 # Se o argumento de backup estiver vazio, dá erro
30 if [[ $BACKUP_DIR == '' ]]; then
31     echo "Uso: $0 [-c] <src> <backup_dst>"
32     exit 1
33 fi
```

Após todas as verificações, começamos por dar *echo* do comando que seria executado para criar o diretório de backup, só executando o comando se a opção -c não for fornecida.

```
34
35 # Cria o diretório de destino se não existir
36 if [[ ! -d "$BACKUP_DIR" ]]; then
37   echo -e "mkdir '$BACKUP_DIR'\n"
38
39   # Se não estiver em checking cria a pasta
40   if [[ $CHECK_MODE =/= "-c" ]]; then
41     mkdir -p "$BACKUP_DIR"
42   fi
43 fi
44
```

Por fim, temos um loop que vai percorrer todo o diretório de trabalho, mostrando, tal como o bloco anterior, os comandos que seriam executados para realizar a cópia dos ficheiros para o diretório backup, apenas executando os mesmos se a opção -c não for dada como argumento

```
# Percorre todos os arquivos no diretório de origem
for FILE in "$SRC_DIR"/*; do
  # Verifica se é um ficheiro (não diretório)
  if [[ -f "$FILE" ]]; then
    # Guarda o caminho do ficheiro
    BACKUP_FILE="$BACKUP_DIR/$(basename "$FILE")"
    # Verifica se o ficheiro já existe ou, se exstir, a data de modificação.
    if [[ ! -f "$BACKUP_FILE" || "$FILE" -nt "$BACKUP_FILE" ]]; then
      echo "cp -a '$FILE' '$BACKUP_FILE'"
      # Se não estiver em checking cria cópias dos ficheiros
      if [[ "$CHECK_MODE" != "-c" ]]; then
        cp -a "$FILE" "$BACKUP_FILE"
      fi
    fi
  fi
done
```

2.2 Testes

Neste primeiro teste realizámos o backup da pasta, mostrando que o comando -c não executa os comandos quando usado e ainda que, após o backup nenhum comando é executado, em nenhum dos casos.

```
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$ ./backup_files.sh -c test_src/ test_bck/
mkdir 'test_bck/'
cp -a 'test_src//a\n.txt' 'test_bck//a\n.txt'
cp -a 'test_src//a.txt' 'test_bck//a.txt'
cp -a 'test_src//ficheiro nome grande e espaços.txt' 'test_bck//ficheiro nome grande e espaços.txt'
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$ ./backup_files.sh test_src/ test_bck/
mkdir 'test_bck/'
cp -a 'test_src//a\n.txt' 'test_bck//a\n.txt'
cp -a 'test_src//a.txt' 'test_bck//a.txt'
cp -a 'test_src//ficheiro nome grande e espaços.txt' 'test_bck//ficheiro nome grande e espaços.txt'
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$ ./backup_files.sh -c test_src/ test_bck/
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$ ./backup_files.sh test_src/ test_bck/
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$
```

Neste segundo teste alterámos um ficheiro na pasta source após o backup e realizámos o backup mais uma vez, onde se verificou que o ficheiro foi atualizado na pasta backup.

```
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$ ./backup_files.sh -c test_src/ test_bck/
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$ ./backup_files.sh test_src/ test_bck/
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$ ./backup_files.sh test_src/ test_bck/
cp -a 'test_src//a.txt' 'test_bck//a.txt'
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$
```

3 Backup

3.1 Implementação

No ficheiro **backup.sh** começámos por criar uma função para exibir o uso correto da chamada do ficheiro sem ter que repetir blocos de código, desta vez com opções a mais, também opcionais, sendo estas [-b tfile] e [-r regexpr].

```
#!/bin/bash

# Função para dar print do comando correto caso haja algum erro
function usage() {
  echo "Uso: $0 [-c] [-b tfile] [-r regexpr] <SRC_DIR> <BACKUP_DIR>"
  exit 1
}
```

Também fizemos mais uma função, desta vez uma função para verificação da expressão regular fornecida na opção -r. Esta função retorna erro caso a *regular expresion* fornecida não seja válida.

```
# Função de verificação do regex
function regexCheck() {

        if [[ "" =~ $REGEX ]]; then
          echo ""
        else
        # Quando o registo dá código 2, dá o regex como inválido
        if [[ $? -eq 2 ]]; then
          echo "Regex inválido: $REGEX"
          exit 1
        fi
      fi
}


```

Aqui procedemos, tal como no primeiro script, à inicialização das variáveis correspondentes aos argumentos fornecidos e ao *array* onde irão estar armazenados os nomes dos ficheiros a ignorar.

```
# Inicialização das variáveis relativas aos argumentos -b, -r, -c
CHECK_MODE=""
TEXT_FILE=""
REGEX=""

# Cria um array para armazenar os nomes dos arquivos de exceção
declare -a EXCEPTION_FILES=()
```

Neste script, pelo facto de haver mais do que um argumento possível, optámos por utilizar o ***getopts***, que faz a verificação da forma correta para usar cada um dos argumentos, para além de atribuir o valor fornecido às variáveis correspondentes a cada uma das opções. Por fim, deixamos apenas os argumentos dos diretórios, para os atribuir às variáveis e fazer as verificações que foram mostradas no script anterior.

```
# Usa getopts para selecionar os
while getopts "cb:r:" opt; do
  case "$opt" in
  c)
    # Opção c (caso chamada, não copia os ficheiros)
    CHECK_MODE="-c"
    ;;
  b)
    # Opção -b (lê o ficheiro e coloca cada linha num array)
    TEXT_FILE="$OPTARG"

    if [[ ! -f "$TEXT_FILE" ]]; then
      echo "O ficheiro '$TEXT_FILE' não existe."
      TEXT_FILE=""
      usage
    elif [[ -n "$TEXT_FILE" ]]; then
      {
        while IFS= read -r LINE || [ -n "$LINE" ]; do
          EXCEPTION_FILES+=("$LINE")
        done
      } < "$TEXT_FILE"
    fi
    ;;
  r)
    # Opção -r (usa um regex e vê se o mesmo é válido)
    REGEX="$OPTARG"
    regexCheck
    ;;
  *)
    # Caso tenha um argumento diferente de -c, -b, -r, dá erro de argumento inválido
    echo "Argumento inválido: -$opt"
    usage
    ;;
  esac

done

# Remove os argumentos e deixa apenas os dois diretórios
shift $((OPTIND - 1))
```

Nesta última parte do script, alterámos o código do script anterior de modo a verificar, através das variáveis das opções, se tem ficheiros para ignorar. Apenas irá ignorar os ficheiros que **não estiverem de acordo com a regular expression (dada na opção -r** ou **que tenham o nome dentro do ficheiro de texto (dado na opção -b)**. Para além disso, adicionámos a parte onde verifica se tem subdiretórios e, caso tenha, chama recursivamente o script para fazer o backup dos mesmos, tal como dos ficheiros que se encontram neles.

```
# Percorre todos os arquivos no diretório de origem
for FILE in "$SRC_DIR"/*; do
  BASENAME=$(basename "$FILE")

  # Verifica se o ficheiro está na lista de exceções
  if [[ " ${EXCEPTION_FILES[*]} " == *" $BASENAME "* ]]; then

    if [[ -d "$FILE" ]]; then
      TARGET_DIR="$BACKUP_DIR/$BASENAME"
      echo "mkdir '$TARGET_DIR'"
      if [[ "$CHECK_MODE" != "-c" ]]; then
        mkdir -p "$TARGET_DIR"
      fi
    fi
    continue
  fi

  # Verifica se é ficheiro e se corresponde ao regex fornecido
  if [[ -f "$FILE" && ( -z "$REGEX" || "$BASENAME" =~ $REGEX ) ]]; then

    BACKUP_FILE="$BACKUP_DIR/$BASENAME"
    # Verifica se é ficheiro e se não se encontra na lista de ficheiros a excluír
    if [[ ! -f "$BACKUP_FILE" || "$FILE" -nt "$BACKUP_FILE" ]]; then
      echo "cp -a '$FILE' '$BACKUP_FILE'"
      if [[ "$CHECK_MODE" != "-c" ]]; then
        cp -a "$FILE" "$BACKUP_FILE"
      fi
    fi

  # Para os subdiretórios, chama a função recursivamente e com os mesmos argumentos

  elif [[ -d "$FILE" ]]; then

    CMD=(bash "$0")
    [[ -n "$CHECK_MODE" ]] && CMD+=("-c")
    [[ -n "$TEXT_FILE" ]] && CMD+=("-b" "$TEXT_FILE")
    [[ -n "$REGEX" ]] && CMD+=("-r" "$REGEX")
    CMD+=("$FILE" "$BACKUP_DIR/$BASENAME")
    "${CMD[@]}"
  fi
done
```

3.2 Testes

O primeiro teste que fizemos nesta iteração foi o teste do backup normal (apenas com o argumento de checking, para comparação), seguido de um teste com a opção -b sem e com ficheiro de texto, dando erro no primeiro teste e ignorando alguns ficheiros no segundo.

```
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$ ./backup.sh -c test_src test_bck
mkdir 'test_bck'
cp -a 'test_src/a\n.txt' 'test_bck/a\n.txt'
cp -a 'test_src/a.txt' 'test_bck/a.txt'
mkdir 'test_bck/c'
mkdir 'test_bck/c/cantcopy'
cp -a 'test_src/c/c.txt' 'test_bck/c/c.txt'
mkdir 'test_bck/c/teste'
cp -a 'test_src/ficheiro nome grande e espaços.txt' 'test_bck/ficheiro nome grande e espaços.txt'
mkdir 'test_bck/p1'
cp -a 'test_src/p1/a.asm' 'test_bck/p1/a.asm'
mkdir 'test_bck/p1/abc'
cp -a 'test_src/p1/abc/abc.m' 'test_bck/p1/abc/abc.m'
cp -a 'test_src/p1/abc/a.p' 'test_bck/p1/abc/a.p'
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$ ./backup.sh -c -b test_src test_bck
O ficheiro 'test_src' não existe.
Uso: ./backup.sh [-c] [-b tfile] [-r regexpr] <SRC_DIR> <BACKUP_DIR>
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$ ./backup.sh -c -b optb_file.txt test_src test_bck
mkdir 'test_bck'
cp -a 'test_src/a\n.txt' 'test_bck/a\n.txt'
cp -a 'test_src/a.txt' 'test_bck/a.txt'
mkdir 'test_bck/c'
cp -a 'test_src/ficheiro nome grande e espaços.txt' 'test_bck/ficheiro nome grande e espaços.txt'
mkdir 'test_bck/p1'
cp -a 'test_src/p1/a.asm' 'test_bck/p1/a.asm'
mkdir 'test_bck/p1/abc'
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$
```

Para testar a opção -r, usámos um regex inválido, seguido do regex que filtra os ficheiros que começam por "a". O primeiro teste, tal como devia, terminou o programa com um erro de regex inválido e o segundo teste apenas copiou os ficheiros que obedeciam à expressão regular.

```
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$ ./backup.sh -c -r "\\" test_src test_bck
Regex inválido: \
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$ ./backup.sh -c -r "^a" test_src test_bck
mkdir 'test_bck'
cp -a 'test_src/a\n.txt' 'test_bck/a\n.txt'
cp -a 'test_src/a.txt' 'test_bck/a.txt'
mkdir 'test_bck/c'
mkdir 'test_bck/c/cantcopy'
mkdir 'test_bck/c/teste'
mkdir 'test_bck/p1'
cp -a 'test_src/p1/a.asm' 'test_bck/p1/a.asm'
mkdir 'test_bck/p1/abc'
cp -a 'test_src/p1/abc/abc.m' 'test_bck/p1/abc/abc.m'
cp -a 'test_src/p1/abc/a.p' 'test_bck/p1/abc/a.p'
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$
```

Neste teste verificámos tanto a compatibilidade de todas as opções ao mesmo tempo, como a permutação das mesmas.

```
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$ ./backup.sh -r "^f" -c -b optb_file.txt test_src/ test_bck/
mkdir 'test_bck/'
mkdir 'test_bck//c'
cp -a 'test_src//ficheiro nome grande e espaços.txt' 'test_bck//ficheiro nome grande e espaços.txt'
mkdir 'test_bck//p1'
mkdir 'test_bck//p1/abc'
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$ ./backup.sh -b optb_file.txt -r "^f" -c test_src/ test_bck/
mkdir 'test_bck/'
mkdir 'test_bck//c'
cp -a 'test_src//ficheiro nome grande e espaços.txt' 'test_bck//ficheiro nome grande e espaços.txt'
mkdir 'test_bck//p1'
mkdir 'test_bck//p1/abc'
rafael@Rafael:~/OneDrive/uni/2º Ano/1º Semestre/SO/proj1$
```

4 Backup Summary

4.1 Implementação

Nesta iteração, as alterações começaram pela encapsulação da lógica principal do backup.sh na função **backup_files()**. Isso permite que a operação de backup seja feita de forma recursiva, para conseguir processar subdiretórios sem duplicação de código.

```
function backup_files() {
  local src_dir="$1"
  local backup_dir="$2"

  # Contadores locais para cada diretório
  local dir_file_copy=0
  local dir_file_update=0
  local dir_file_deleted=0
  local dir_warnings=0
  local dir_errors=0
  local dir_size_copied=0
  local dir_size_deleted=0

  # Faz com que a opção /* no loop fique vazia, caso necessário, para evitar erros em caso de diretórios vazios
  shopt -s nullglob

  # Primeiro, processa os arquivos no diretório atual
  for FILE in "$src_dir"/*; do
    if [[ -f "$FILE" && ( -z "$REGEX" || "$(basename "$FILE")" =~ $REGEX ) ]]; then
      FILENAME=$(basename "$FILE" | cut -d. -f1)

      # Ignora arquivos na lista de exceções
      if [[ " ${EXCEPTION_FILES[*]} " == *" $FILENAME "* ]]; then
        continue
      fi

      local backup_file="$backup_dir/$(basename "$FILE")"

      local file_size=$(stat -c%s "$FILE")

      # Se o arquivo não existe no backup, copia e incrementa o contador e tamanho
      if [[ ! -f "$backup_file" ]]; then
        echo "cp -a $FILE $backup_file"
        dir_file_copy=$((dir_file_copy + 1))
        dir_size_copied=$((dir_size_copied + file_size))
        if [[ "$CHECK_MODE" != "-c" ]]; then
          cp -a "$FILE" "$backup_file"
        fi

        # Se der erro na cópia, incrementa o contador de erros
        if [[ $? -ne 0 ]]; then
          echo "ERROR: Failed to copy '$FILE' to '$backup_file'"
          dir_errors=$((dir_errors + 1))
        fi


      ...
```

A função dispõe também de contadores locais para erros, avisos e o número de ficheiros modificados, copiados e eliminados, tal como o espaço ocupado / libertado nas operações descritas. Estes contadores são mostrados num sumário que, para cada pasta dentro do diretório, mostra várias informações pertinentes. Assim sendo, tiveram de ser implementados incrementadores sempre que alguma dessas ações existem, bem como a utilização do return value (**$?**) das funções para contabilizar erros.

Foi adicionado código que permite eliminar ficheiros que estejam no diretório de backup e não estejam no diretório de trabalho, tal como o método **shopt**, que permite evitar erros ao processar diretórios vazios.

```
111  ...
112
113   # Verifica se o diretório de backup existe: se estiver em check e o diretório de backup não existir, não h
    á ficheiros para copiar
114   if [[ -d "$backup_dir" ]]; then
115     # Remove arquivos do backup que não estão no diretório de origem
116     for BACKUP_FILE in "$backup_dir"/*; do
117       local src_file="$src_dir/$(basename "$BACKUP_FILE")"
118       if [[ ! -e "$src_file" ]]; then
119         local backup_file_size=$(stat -c%s "$BACKUP_FILE")
120         dir_file_deleted=$((dir_file_deleted + 1))
121         dir_size_deleted=$((dir_size_deleted + backup_file_size))
122         if [[ "$CHECK_MODE" ≠ "-c" ]]; then
123           rm -rf "$BACKUP_FILE"
124         fi
125       fi
126     done
127   fi
128
129   # Fecha a verificação
130   shopt -u nullglob
131
132   # Atualiza os contadores globais com os valores do diretório
133   TOTAL_FILE_COPY=$((TOTAL_FILE_COPY + dir_file_copy))
134   TOTAL_FILE_UPDATE=$((TOTAL_FILE_UPDATE + dir_file_update))
135   TOTAL_FILE_DELETED=$((TOTAL_FILE_DELETED + dir_file_deleted))
136   TOTAL_WARNINGS=$((TOTAL_WARNINGS + dir_warnings))
137   TOTAL_ERRORS=$((TOTAL_ERRORS + dir_errors))
138   TOTAL_SIZE_COPIED=$((TOTAL_SIZE_COPIED + dir_size_copied))
139   TOTAL_SIZE_DELETED=$((TOTAL_SIZE_DELETED + dir_size_deleted))
140
141   # Exibe o resumo de operações para o diretório atual
142   if [[ CHECK_MODE ≠ "-c"  ]]; then
143     echo -e "While backuping $src_dir: $dir_errors Errors; $dir_warnings Warnings; $dir_file_update Updated;
    $dir_file_copy Copied (${dir_size_copied}B); $dir_file_deleted Deleted (${dir_size_deleted}B)"
144   else
145     echo -e "While backuping $src_dir: $dir_file_update Updated; $dir_file_copy Copied (${dir_size_copied}
    B); $dir_file_deleted Deleted (${dir_size_deleted}B)"
146   fi
147 }
```

Fora da função, foram criadas formas de lidar com casos de erros específicos, com os quais é impossível a execução do programa. Os casos estudados foram os seguintes:

- 1: se o diretório de origem dos ficheiros não existir;
- 2: se o diretório de backup não tiver sido expecificado;
- 3: se o diretório de backup não existir e não for possível criar a pasta;
- 4: se o espaço disponível no diretório de backup for insuficiente;
- 5: se o diretório de backup é um subdiretório do diretório de trabalho.

Se algum destes erros for identificado, uma nova variável **FLAG_ERROR** mudará o seu valor e será esta a responsável por não deixar a função de backup ser executada. Serão mostradas mensagens para cada um dos erros cometidos e, com a implementação de contadores globais, será mostrado um sumário com as mesmas informações que as existentes dentro da função **backup_files()**. Apesar de estarem completamente implementados, os valores destes contadores globais só serão mostrados no terminal no caso de qualquer um dos erros descritos anteriormente for detetado.

```
188
189 # Inicialização dos contadores globais
190 TOTAL_FILE_COPY=0
191 TOTAL_FILE_UPDATE=0
192 TOTAL_FILE_DELETED=0
193 TOTAL_WARNINGS=0
194 TOTAL_ERRORS=0
195 TOTAL_SIZE_COPIED=0
196 TOTAL_SIZE_DELETED=0
197
198 FLAG_ERROR=0
199
200 # Verifica se o diretório de origem existe
201 if [[ ! -d "$SRC_DIR" ]]; then
202   echo "ERROR: The source directory '$SRC_DIR' does not exist."
203   TOTAL_ERRORS=$((TOTAL_ERRORS + 1))
204   FLAG_ERROR=1
205 fi
206
207 # Verifica se o diretório de backup foi especificado
208 if [[ $BACKUP_DIR == '' ]]; then
209   TOTAL_ERRORS=$((TOTAL_ERRORS + 1))
210   FLAG_ERROR=1
211 fi
212
213 # Cria o diretório de destino, se não existir
214 if [[ ! -d "$BACKUP_DIR" ]]; then
215   echo "mkdir $BACKUP_DIR"
216   if [[ $CHECK_MODE ≠ "-c" ]]; then
217     mkdir -p "$BACKUP_DIR"
218   fi
219
220   # Se houver erro ao criar o diretório de backup, exibe mensagem de erro
221   if [[ $? -ne 0 ]]; then
222     echo "ERROR: Failed to create the backup directory: $BACKUP_DIR"
223     TOTAL_ERRORS=$((TOTAL_ERRORS + 1))
224     FLAG_ERROR=1
225   fi
226 fi
227
228 # Verifica se SRC_DIR e BACKUP_DIR são diretórios
229 if [[ -d "$SRC_DIR" && -d "$BACKUP_DIR" ]]; then
230
231   # Obtém o caminhos completos
232   FULL_SRC_DIR=$(realpath "$SRC_DIR")
233   FULL_BACKUP_DIR=$(realpath "$BACKUP_DIR")
234
235   # Obtém o diretório pai de SRC_DIR
236   PARENT_BACKUP_DIR=$(dirname "$FULL_BACKUP_DIR")
237
238   # Obtém o tamanho do diretório de origem
239   SRC_SIZE=$(du -s "$SRC_DIR" | awk '{print $1}')
240   # Obtém o espaço disponível no diretório de backup
241   BACKUP_FREE=$(df "$PARENT_BACKUP_DIR" | awk 'NR==2 {print $4}')
242
243   # Verifica se há armazenamento suficiente no diretório de backup
244   if [[ "$SRC_SIZE" -gt "$BACKUP_FREE" ]]; then
245     echo "ERROR: backup directory does not have enough space"
246     TOTAL_ERRORS=$((TOTAL_ERRORS + 1))
247     FLAG_ERROR=1
248   fi
249 fi
250
251 # Verifica se FULL_SRC_DIR é parte de FULL_BACKUP_DIR
252 if [[ "$FULL_BACKUP_DIR" == "$FULL_SRC_DIR" || "$FULL_BACKUP_DIR" == "$FULL_SRC_DIR/"* ]]; then
253   echo "ERROR: The backup directory cannot be a subdirectory of the source directory."
254   TOTAL_ERRORS=$((TOTAL_ERRORS + 1))
255   FLAG_ERROR=1
256 fi
257
258 # Executa a função de backup na raiz se não houver erros que não o permitam
259 if [[ $FLAG_ERROR -eq 0 ]]; then
260   backup_files "$SRC_DIR" "$BACKUP_DIR"
261 else
262     echo -e "\nWhile backuping $SRC_DIR: $TOTAL_ERRORS Errors; $TOTAL_WARNINGS Warnings; $TOTAL_FILE_UPDATE Updated;
    $TOTAL_FILE_COPY Copied ($TOTAL_SIZE_COPIED B); $TOTAL_FILE_DELETED Deleted($TOTAL_SIZE_DELETED B);"
263 fi
264
```

4.2 Testes

O primeiro teste que fizemos nesta iteração foi o teste do backup normal. De seguida, foi eliminado um ficheiro no diretório de trabalho, para testar se, após correr o programa, o ficheiro correspondente no diretório de backup foi também eliminado. Por fim, modificámos um ficheiro no diretório de backup, para verificar se é dado um aviso ao voltar a correr o programa.

```
rodrigo@Legion5-RS:~/Desktop/Uni-21/SO/proj1$ ./backup_summary.sh ./test_src/ ./test_backup
mkdir ./test_backup
cp -a ./test_src//a\n.txt ./test_backup/a\n.txt
cp -a ./test_src//a.txt ./test_backup/a.txt
cp -a ./test_src//ficheiro nome grande e espaços.txt ./test_backup/ficheiro nome grande e espaços.txt
mkdir ./test_backup/c
cp -a ./test_src//c/c.txt ./test_backup/c/c.txt
While backuping ./test_src//c: 0 Errors; 0 Warnings; 0 Updated; 1 Copied (112B); 0 Deleted (0B)
mkdir ./test_backup/p1
cp -a ./test_src//p1/a.asm ./test_backup/p1/a.asm
mkdir ./test_backup/p1/abc
cp -a ./test_src//p1/abc/abc.m ./test_backup/p1/abc/abc.m
cp -a ./test_src//p1/abc/a.p ./test_backup/p1/abc/a.p
While backuping ./test_src//p1/abc: 0 Errors; 0 Warnings; 0 Updated; 2 Copied (0B); 0 Deleted (0B)
While backuping ./test_src//p1: 0 Errors; 0 Warnings; 0 Updated; 1 Copied (63B); 0 Deleted (0B)
While backuping ./test_src/: 0 Errors; 0 Warnings; 0 Updated; 3 Copied (267B); 0 Deleted (0B)
rodrigo@Legion5-RS:~/Desktop/Uni-21/SO/proj1$ ./backup_summary.sh ./test_src/ ./test_backup
While backuping ./test_src//c: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
rm -rf ./test_backup/p1/abc/abc.m
While backuping ./test_src//p1/abc: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 1 Deleted (0B)
While backuping ./test_src//p1: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
While backuping ./test_src/: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
rodrigo@Legion5-RS:~/Desktop/Uni-21/SO/proj1$ ./backup_summary.sh ./test_src/ ./test_backup
WARNING: './test_backup/a\n.txt' was modified more recently than './test_src//a\n.txt'
While backuping ./test_src//c: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
While backuping ./test_src//p1/abc: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
While backuping ./test_src//p1: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
While backuping ./test_src/: 0 Errors; 1 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
rodrigo@Legion5-RS:~/Desktop/Uni-21/SO/proj1$
```

Para testar os erros que inviabilizam a continuidade do projeto, começámos por tentar fazer backup sem indicar nenhum dos diretórios necessários. De seguida, corremos o programa, tendo como diretório de destino um dispositivo amovível com capacidade inferior ao tamanho total do diretório de trabalho. Por fim, temos o caso de tentar correr o programa indicando uma subpasta do diretório de trabalho como diretório de backup. Em todos os casos, o espectável é dar uma mensagem de erro por cada erro que ocorrer e terminar de correr o programa com um sumário que contém a contagem de erros.

```
rodrigo@Legion5-RS:~/Desktop/Uni-21/SO/proj1$ ./backup_summary.sh -c
ERROR: The source directory '' does not exist.
mkdir
ERROR: The backup directory cannot be a subdirectory of the source directory.

While backuping : 3 Errors; 0 Warnings; 0 Updated; 0 Copied (0 B); 0 Deleted(0 B);
rodrigo@Legion5-RS:~/Desktop/Uni-21/SO/proj1$
rodrigo@Legion5-RS:~/Desktop/Uni-21/SO/proj1$ ./backup_summary.sh -c ~/Desktop/  /media/rodrigo/USB\ STICK/
ERROR: backup directory does not have enough space

While backuping /home/rodrigo/Desktop/: 1 Errors; 0 Warnings; 0 Updated; 0 Copied (0 B); 0 Deleted(0 B);
rodrigo@Legion5-RS:~/Desktop/Uni-21/SO/proj1$
rodrigo@Legion5-RS:~/Desktop/Uni-21/SO/proj1$ ./backup_summary.sh -c test_src/ test_src/c/
ERROR: The backup directory cannot be a subdirectory of the source directory.

While backuping test_src/: 1 Errors; 0 Warnings; 0 Updated; 0 Copied (0 B); 0 Deleted(0 B);
rodrigo@Legion5-RS:~/Desktop/Uni-21/SO/proj1$
```

5 Backup Check

5.1 Implementação

Iniciámos o desenvolvimento desta última iteração da mesma forma que nos outros ficheiros, atribuindo os caminhos para os diretórios às variáveis correspondentes e verificando a existência dos mesmos. Após isso, temos um *loop* que, através do método **md5sum** verifica se os ficheiros cujo nome é igual têm diferenças entre si (guarda os hashes de ambos os ficheiros e compara-os no final) e, caso encontre uma pasta, chama o script recursivamente.

```bash
#!/bin/bash

# Diretórios de origem e destino
SRC_DIR="$1"
DEST_DIR="$2"

# Verifica se o diretório de origem existe
if [[ ! -d "$SRC_DIR" ]]; then
  echo "Diretório de origem não existe: $SRC_DIR"
  exit 1
fi

# Verifica se o diretório de destino existe
if [[ ! -d "$DEST_DIR" ]]; then
  echo "Diretório de destino não existe: $DEST_DIR"
  exit 1
fi

# Itera sobre todos os arquivos no diretório de origem
for FILE in "$SRC_DIR"/*; do

  # Construi o caminho completo para o arquivo no diretório de destino
  DEST_FILE="$DEST_DIR/$(basename "$FILE")"

  # Verifica se o arquivo existe no diretório de destino
  if [[ -f "$DEST_FILE" ]]; then

    SRC_FILE="$FILE"

    # Calcula os hashes MD5 dos dois arquivos
    SRC_HASH=$(md5sum "$SRC_FILE" | cut -d' ' -f1)
    DEST_HASH=$(md5sum "$DEST_FILE" | cut -d' ' -f1)

    # Compara os hashes e imprime a mensagem de erro se forem diferentes
    if [[ "$SRC_HASH" != "$DEST_HASH" ]]; then
      echo ""$SRC_FILE" "$DEST_FILE" differ."
    fi

  fi

  if [[ -d "$DEST_FILE" ]]; then
    bash "$0" "$SRC_DIR/$(basename "$DEST_FILE")" "$DEST_FILE"
  fi

done
```

5.2 Testes

Por fim realizámos o backup e logo em seguida executámos o script. Após isto, alterámos dois ficheiros, um no diretório principal e outro num subdiretório do diretório de trabalho e ainda removemos um dos ficheiros. Seguidamente voltámos a executar o script e, como se pode verificar pelo output seguinte, no primeiro caso não deu nenhum ficheiro como diferente e no segundo deu output a dois erros, um por cada ficheiro modificado.

```
rafael@Rafael:~/Desktop/uni/2º Ano/1º Semestre/SO/proj1$ ./backup_check.sh test_src test_bck
rafael@Rafael:~/Desktop/uni/2º Ano/1º Semestre/SO/proj1$ ./backup_check.sh test_src test_bck
test_src/a.txt test_bck/a.txt differ.
test_src/p1/abc/a.p test_bck/p1/abc/a.p differ.
rafael@Rafael:~/Desktop/uni/2º Ano/1º Semestre/SO/proj1$
```

6 Conclusão

Neste projeto, os requisitos que foram indicados no enunciado foram corretamente implementados, dado que os resultados coincidem com o que era esperado.

Durante o processo de implementação, encontrámos obstáculos/problemas em diferentes etapas do projeto. Estes desafios foram superados pela aplicação de conhecimentos adquiridos nas aulas práticas e teóricas, tal como pesquisa complementar feita na internet.

Este projeto foi muito útil para consolidação dos conceitos da linguagem bash e para melhorar, de certa forma, a nossa habilidade para utilizar o terminal em sistemas Unix.

7 Bibliografia

[1] StackOverflow: `https://stackoverflow.com`

[2] GNU Bash Manual: `https://www.gnu.org/software/bash/manual/html_node/index.html#SEC_Contents`

[3] Man Pages - Bash: `https://man.cx/bash`

[4] Educative.io: `https://www.educative.io`

[5] Cheatsheets.zip: `https://cheatsheets.zip`