universidade de aveiro deti departamento de electrónica, telecomunicações e informática

# Relatório do Trabalho 1

## Ferramenta de criação/atualização de cópias de segurança em bash

## Sistemas Operativos

Professor:
José Nuno Panelas Nunes Lau
nunolau@ua.pt

**Trabalho Realizado Por:**
Eduardo José Farinha do Rosário nº119234
José Pedro da Costa Bagagem nº120141

# Índice

# 1 Introdução

No âmbito da unidade curricular de Sistemas Operativos, apresentamos o primeiro trabalho desta disciplina. Este projeto tem como objetivo desenvolver um script em Bash que permita criar e atualizar uma cópia de segurança de um diretório. Para atingir este objetivo, o projeto foi dividido em quatro scripts, sendo os três primeiros iterações que vão elaborando progressivamente a versão anterior até chegar ao objetivo final. O último script serve para verificar se o conteúdo dos ficheiros do diretório de backup corresponde aos ficheiros do diretório de trabalho.

Ao longo deste relatório, explicaremos os raciocínios usados durante a realização destes scripts, bem como as adaptações feitas nos códigos de scripts anteriores para resolver problemas maiores. Para além disso, mencionaremos alguns dos erros e exceções encontrados durante o desenvolvimento do projeto e quais as soluções adotadas para os resolver.

Posteriormente, abordaremos os vários testes realizados para validar as soluções implementadas.

# 2 Primeira Iteração - backup_files.sh

O script `backup_files.sh` é uma versão simplificada do projeto final, começando assim por resolver um problema mais simples para depois poder expandir o programa. Por isso, consideramos que a diretoria de trabalho teria apenas arquivos.

Este script terá apenas uma opção, pois esta é essencial para facilitar o processo de testagem do programa:

**Opção -c**: O script imprime no terminal exatamente o mesmo que iria imprimir se esta opção não fosse selecionada, mas não cria nem atualiza qualquer ficheiro.

Para além disso, o script ainda tem de ter obrigatoriamente os argumentos:

**Diretório de trabalho**: Argumento que especifica o caminho absoluto ou relativo do diretório do qual se deseja criar uma cópia de segurança.

**Diretório de backup**: Argumento que especifica o caminho absoluto ou relativo do diretório onde será guardada a cópia de segurança.

## 2.1 Inicialização das variáveis globais

O programa começa por inicializar as variáveis globais, por uma questão de clareza. Os valores atribuídos não são definitivos na maioria dos casos, servindo apenas para inicializar as variáveis. A variável `CHECKING` indica se a opção -c foi selecionada, sendo 1 quando é o caso. A variável `WorkDir` indica o diretório de trabalho e a `Backup` indica o diretório de backup.

```
CHECKING="0"
WORKDIR=""
BACKUP=""
```

Figura 1: Inicialização das variáveis globais

## 2.2 Verificação das Opções

Após inicializar as variáveis globais, o programa irá verificar se alguma opção foi selecionada, que neste caso é apenas a opção -c. Sendo assim, a variável `CHECKING` muda para 1 quando a opção de *checking* for selecionada. Se for selecionada uma opção que não exista, o programa imprime uma mensagem de erro e para a execução do programa. No final, retiramos os parâmetros opcionais da lista de argumentos, visto que já temos o que precisamos sobre este guardado.

```
while getopts ":c" opt; do
    case $opt in
        c)
            CHECKING="1"
            ;;
        \?)
            echo "ERROR: -$OPTARG is an invalid option"
            exit 1
            ;;
    esac
done

# Move a posição dos parâmetros da linha de comando após o uso do getopts
shift $((OPTIND - 1))
```

Figura 2: Código para verificação das opções do script

## 2.3 Validação dos caminhos dos argumentos

Para continuar o projeto é necessário saber qual o diretório de trabalho e o de backup, mas para isso precisamos de fazer várias validações. Primeiramente, pensamos na possibilidade de o utilizador não escrever os dois argumentos obrigatórios, para isso, verificámos, após retirar os argumentos opcionais da lista, se o número de argumentos é igual a dois, caso contrário, o programa imprime uma mensagem de erro e termina. Seguidamente, verificámos se o diretório de trabalho existe, para depois guardarmos na nossa variável global `WorkDir` o seu caminho absoluto. A seguir, usamos a função `mkdirprint()`, que iremos explicar mais à frente, mas neste caso, ela cria o diretório de backup e imprime o comando, se esta ainda não existir. Por fim, guardamos o caminho absoluto do backup e verificamos se não é igual ao caminho do diretório que vamos copiar, visto que, se assim acontecer, não fará sentido executar o programa.

```
1  if [[ $# -eq 2 ]]; then
2      echo "ERROR: Not enough arguments"
3      exit 1
4  elif [[ ! -d "$1" ]]; then
5      echo "ERROR: "$1" not a directory"
6      exit 1;
7  fi
8  WORDIR="$(realpath "$1")"
9
10 mkdirprint "$2" "$2"
11
12 BACKUP="$(realpath "$2")"
13 if [[ "$BACKUP" == "$WORKDIR" ]]; then
14     echo "ERROR: The arguments are equal"
15     exit 1
16 fi
```

Figura 3: Validação dos caminhos dos argumentos

### 2.3.1 Validação das permissões dos diretórios

Se os diretórios não tiverem certas permissões, o programa também irá ter problemas na sua execução. Ou seja, se o diretório de trabalho não tiver permissão de leitura, então o script não irá conseguir aceder aos ficheiros dentro dele e por isso não consegue fazer a cópia de segurança. Para além disso, se o diretório para onde pretendemos fazer a cópia de trabalho não tiver a permissão de escrita, o programa não irá conseguir copiar os ficheiros. Assim, se algum destes casos acontecer, o script imprime uma mensagem de erro e termina a execução com o valor de retorno um.

```
1  if [[ ! -r "$1" ]]; then
2      echo "ERROR: "$(basename "$WORKDIR")" doesnt have read permissions"
3      exit 1
4  fi
5
6  if [[ -d "$2" ]] && [[ ! -w "$2" ]]; then
7      echo "ERROR: "$(basename "$BACKUP")" doesnt have write permissions"
8      exit 1
9  fi
```

Figura 4: Validação das permissões dos diretórios

### 2.3.2 Validação do Espaço Disponível

Deparámo-nos ainda com a possibilidade do script tentar copiar para um diretório sem espaço suficiente, como, por exemplo, se tentássemos realizar uma cópia de segurança numa pen que já estava cheia. Para isso precisamos então de implementar mais uma verificação. Esta começa por ver o espaço do diretório de trabalho, seguidamente, precisamos de verificar também se o diretório de backup existe, algo que pode não acontecer se a opção de *checking* for usada. Se este não existir, então em vez de se verificar o espaço disponível neste diretório, é verificado o espaço disponível no diretório pai. Por fim, se o espaço disponível for menor do que a do espaço do diretório de trabalho então o script irá imprimir uma mensagem de erro e parar a execução com o valor um.

```
1
2  # Calcula o espaço total de todos os ficheiros no diretório de trabalho (em KB)
3  WorkDirSize=$(du -sk "$WORKDIR" | awk '{print $1}')
4
5  dirToCheck="$BACKUP"
6  if [[ ! -d "$2" ]]; then
7      dirToCheck="$(dirname "$BackupPath")"
8  fi
9
10 # Calcula o espaço disponível para se fazer a cópia (em KB)
11 AvailableSpace=$(df -k "$dirToCheck" | awk 'NR==2 {print $4}')
12
13 if (( AvailableSpace < WorkDirSize )); then
14     echo "ERROR: Not enough space in destination directory."
15     exit 1
16 fi
```

Figura 5: Verificação do espaço disponível

## 2.4 Função mkdirprint()

Quando começámos o projeto percebemos que iríamos ter de criar diretórios e imprimir esses comandos várias vezes e por isso optámos por criar uma função que fizesse isso. A função `mkdirprint()` está noutro ficheiro `utils.sh` onde estão as funções usadas em vários scripts deste projeto. Esta recebe dois argumentos, o primeiro é o diretório que vai ser criado e o segundo é o diretório por onde se começará a imprimir o primeiro argumento.

Como estamos a trabalhar com caminhos absolutos, se imprimíssemos o primeiro argumento da função, íamos ficar com diretórios muito grandes, sendo a maioria da informação desnecessária, então, para aumentar a legibilidade dos *outputs*, optamos por imprimir o caminho relativo do primeiro argumento a partir do segundo argumento.

A função começa por verificar se o diretório existe e se for o caso ela retorna zero. Depois, esta vai buscar o caminho simplificado do primeiro argumento e imprime o comando. Se a opção de *checking* não for selecionada, esta cria o diretório. Caso o diretório seja criado, nós optamos por retornar o valor de retorno de `mkdir`, para que os scripts que usam esta função possam saber se houve algum problema na execução desse comando.

```
function mkdirprint(){
    if [[ -d "$1" ]]; then
        return 0
    fi
    local simpler_name="${1#$(dirname "$2")/}"
    echo "mkdir "$simpler_name""
    if [[ $CHECKING -eq 0 ]]; then
        mkdir "$1";
        return $?;
    fi
    return 0;
}
```

Figura 6: Função `mkdirprint()`

## 2.5 Função cpprint()

Pela mesma razão que criámos a função `mkdirprint()` achámos que deveríamos criar uma função equivalente para o comando `cp`. Esta função também se encontra no ficheiro `utils.sh` e tem dois argumentos, um com o ficheiro original e outro com o caminho para onde se pretende copiar o ficheiro. O comando `cp` é usado com a opção `-a` para que a data de modificação da cópia do ficheiro seja igual ao ficheiro original.

Novamente, nós não queremos imprimir os caminhos absolutos dos ficheiros, sendo desnecessário imprimir a parte do caminho que corresponde à diretoria de trabalho e à de cópia. Assim como o primeiro argumento é um ficheiro que pertence à diretoria de trabalho, então podemos apenas imprimir o caminho do ficheiro a partir do nome da diretoria de trabalho, e a mesma coisa para o segundo argumento, só que este começa a partir do nome da diretoria de backup.

Decidimos dar vários possíveis valores de retorno a esta função, visto que esta pode acabar por fazer coisas diferentes e numa das próximas iterações será preciso saber o que a função realmente fez. Assim os valores de retorno ficaram decididos como zero quando a função copia um ficheiro que não existe no diretório de backup, um quando um ficheiro foi atualizado, dois quando a função não faz nada, três quando é impresso um aviso e quatro quando é impresso uma mensagem de erro.

A função começa por ir buscar os caminhos simplificados dos argumentos e, de seguida, verifica se o ficheiro que se pretende copiar tem permissões de leitura, se não tiver o programa imprime uma mensagem de erro e retorna. Seguidamente, há a verificação da existência de um ficheiro no caminho para onde se pretende copiar, se for o caso, o programa terá de fazer ainda mais verificações. Dentro das verificações necessárias para o ficheiro onde se pretende copiar, uma delas é se esse ficheiro tem permissões de escrita, se este não a tiver, ela também imprime uma mensagem de erro e retorna. Para continuar as verificações, o programa precisa de ir buscar a data de modificação dos dois ficheiros. Se a cópia do ficheiro for mais velha, o programa imprime uma mensagem de aviso e a função retorna. Caso os ficheiros tenham a mesma data de modificação então a função retorna sem imprimir nada. Agora que sabemos que o ficheiro conseguirá ser atualizado, podemos definir a variável `retValue` como 1.

Após estas validações todas, se a função correu até aqui é porque os argumentos estão certos, logo a função apenas imprime o comando de cópia e, se a opção -c não foi selecionada, a função executa o comando que imprimiu. Esta termina retornando a variável `retValue` que poderá ser zero ou um, dependendo da operação realizada.

```
function cpprint(){
    local simpler_name_workdir="${1#$(dirname "$WORKDIR")/}"
    local simpler_name_backup="${2#$(dirname "$BACKUP")/}"
    if [ ! -r "$1" ]; then
        echo "ERROR: "$simpler_name_workdir" doenst have permission to read"
        return 4
    fi
    local retValue="0"
    if [ -f "$2" ]; then
        if [ ! -w "$2" ]; then
            echo "ERROR: "$simpler_name_backup" doenst have permission to write"
            return 4
        fi
        local FILE_MODE_DATE=$(stat -c %Y "$1")
        local BAK_FILE_DATE=$(stat -c %Y "$2")
        if [[ "$FILE_MODE_DATE" -lt "$BAK_FILE_DATE" ]]; then
            echo "WARNING: backup entry $simpler_name_backup is newer than
    $simpler_name_workdir; Should not happen"
            return 3;
        elif [[ "$FILE_MODE_DATE" -eq "$BAK_FILE_DATE" ]]; then
            return 2;
        fi
        retValue="1"
    fi
    echo "cp -a $simpler_name_workdir $simpler_name_backup"
    if [[ $CHECKING -eq 0 ]]; then
        cp -a "$1" "$2";
    fi
    return $retValue
}
```

Figura 7: Função `cpprint()`

## 2.6 Remoção dos ficheiros que não existem no diretório de trabalho

Se o diretório para onde iremos fazer a cópia de segurança já existir, então é provável que esta tenha ficheiros que não pertencem ao diretório de trabalho. Por isso, nós optámos por eliminar primeiro os ficheiros que estão a mais, visto que temos de verificar menos ficheiros no diretório de cópia porque ainda não adicionamos os ficheiros do diretório de trabalho.

Assim nós iteramos por cada ficheiro que existe no diretório de cópia e passamos à frente se for um diretório. De seguida verificamos se ele existe no diretório de trabalho, caso contrário, este será removido, quando a opção de *checking* não for selecionada. Para evitar erros, não apagamos os ficheiros se não tiveram permissão de escrita, passando para a próxima iteração se for o caso.

Ao correr uma versão antiga do script, apercebemo-nos de dois problemas, o primeiro era que o programa não iterava sob ficheiros invisíveis e o segundo era que quando o diretório não existia ou estava vazio o programa iterava uma vez com um próprio padrão. Para evitar isso nós adicionamos a primeira linha do código apresentado em baixo, em que a opção `nullglob` faz com que quando o padrão não corresponde a nenhum ficheiro ele expanda para uma string vazia em vez de expandir para o próprio padrão e a opção `dotglob` faz com que os ficheiros invisíveis também sejam incluídos no *loop*.

```
1  shopt -s nullglob dotglob
2  for file in "$2"/*; do
3      if [[ -d "$file" ]]; then
4          continue;
5      fi
6      if [[ -f "$1/$(basename "$file")" ]]; then
7          continue;
8      fi
9      if [[ ! -w $file_copy ]]; then
10         echo "ERROR: "$(basename "$WORKDIR")/$(basename "$file") doesnt have writing
       permissions""
11         continue;
12     fi
13     if [[ $CHECKING -eq "0" ]]; then
14         rm "$file"
15     fi
16 done
```

Figura 8: Remoção dos ficheiros que não existem no diretório de trabalho

## 2.7 Cópia dos ficheiros

Agora o programa pode finalmente fazer a cópia de segurança. Esta parte do código é como se fosse o inverso da operação anterior, o programa irá então iterar sob cada ficheiro do diretório de trabalho e evocar a função `cpprint()` e, como já tínhamos explicado, esta irá copiar o ficheiro se ele não existir no diretório de cópia ou se for mais antigo do que a versão do diretório de segurança.

Ao iterarmos por cada arquivo, o script ainda verifica primeiro se este não é um diretório. Esta parte não seria necessária, visto que o objetivo era apenas correr este programa em diretórios que apenas tinham ficheiros, mas para conseguirmos testar este programa com mais diretórios, nós decidimos implementar esta condição, visto que esta também não irá mudar os *outputs* nas situações onde esta deve ser executada.

```
1  for file in "$WORKDIR"/*; do
2      if [[ -d "$file" ]]; then
3          continue;
4      fi
5      cpprint "$file" "$BACKUP/$(basename "$file")"
6  done
```

Figura 9: Remoção dos ficheiros que não existem no diretório de trabalho

## 2.8 Testes

Para averiguar a qualidade do script realizámos vários testes diferentes, com o objetivo de garantir que este não tem problemas.

```
~/SO/SO_proj1/source$ ./backup_files.sh ~/SO/aula04
ERROR: The function has two arguments
```

Figura 10: Teste do script sem os dois argumentos de entrada

```
~/SO/SO_proj1/source$ ./backup_files.sh ~/SO/aula04 ~/SO/aula04
ERROR: The arguments are equal
```

Figura 11: Teste do script com os dois argumentos de entrada iguais

```
~/SO/SO_proj1/source$ ~/SO/aula040 .
ERROR: /home/bagagem/SO/aula040 not a directory
```

Figura 12: Teste do script com um diretório de trabalho inexistente

```
~/SO/SO_proj1/source$ chmod -r ~/SO/aula04
~/SO/SO_proj1/source$ ./backup_files.sh -c ~/SO/aula04 ../test
ERROR: aula04 doesnt have read permissions
```

Figura 13: Teste da verificação quando o diretório de trabalho não tem permissões de leitura

```
~/SO/SO_proj1/source$ ./backup_files.sh -t ~/SO/aula04 ../test
ERROR: -t is an invalid option
```

Figura 14: Teste do uso de uma *flag* inexistente

```
~/SO/SO_proj1/source$ mkdir ../test
~/SO/SO_proj1/source$ mkdir ../test chmod -w ../test
~/SO/SO_proj1/source$ ./backup_files.sh ~/SO/aula04 ../test
ERROR: test doesnt have write permissions
```

Figura 15: Teste do diretório de backup sem permissões de escrita

Os próximos testes serão feitos com o seguinte diretório:

```
~/SO/SO_proj1/source$ ls ~/AED/aula02
AED_Guiao_02.pdf  binary_search.c  ex1  ex2  ex2.c  ex3  ex3.c  ex4  ex4.c  ex5  ex5.c
      ex6  ex6.out  ex7  integer_arithmetic_pitfalls.c  primes.c
```

Figura 16: Diretório de teste

```
~/SO/SO_proj1/source$ ./backup_files.sh ~/AED/aula02 ../test
mkdir test
cp -a aula02/AED_Guiao_02.pdf test/AED_Guiao_02.pdf
cp -a aula02/binary_search.c test/binary_search.c
cp -a aula02/ex1 test/ex1
cp -a aula02/ex2 test/ex2
cp -a aula02/ex2.c test/ex2.c
cp -a aula02/ex3 test/ex3
cp -a aula02/ex3.c test/ex3.c
cp -a aula02/ex4 test/ex4
cp -a aula02/ex4.c test/ex4.c
cp -a aula02/ex5 test/ex5
cp -a aula02/ex5.c test/ex5.c
cp -a aula02/ex6 test/ex6
cp -a aula02/ex6.out test/ex6.out
cp -a aula02/ex7 test/ex7
cp -a aula02/integer_arithmetic_pitfalls.c test/integer_arithmetic_pitfalls.c
cp -a aula02/primes.c test/primes.c
~/SO/SO_proj1/source$ ls ../test
AED_Guiao_02.pdf  binary_search.c  ex1  ex2  ex2.c  ex3  ex3.c  ex4  ex4.c  ex5  ex5.c
      ex6  ex6.out  ex7  integer_arithmetic_pitfalls.c  primes.c
~/SO/SO_proj1/source$ ./backup_files.sh ~/AED ../test
~/SO/SO_proj1/source$
```

Figura 17: Teste de uma cópia de segurança para um diretório que não exista

```
mkdir ../test
~/SO/SO_proj1/source$ ./backup_files.sh -c ~/AED/aula02 ../test
cp -a aula02/AED_Guiao_02.pdf test/AED_Guiao_02.pdf
cp -a aula02/binary_search.c test/binary_search.c
cp -a aula02/ex1 test/ex1
cp -a aula02/ex2 test/ex2
cp -a aula02/ex2.c test/ex2.c
cp -a aula02/ex3 test/ex3
cp -a aula02/ex3.c test/ex3.c
cp -a aula02/ex4 test/ex4
cp -a aula02/ex4.c test/ex4.c
cp -a aula02/ex5 test/ex5
cp -a aula02/ex5.c test/ex5.c
cp -a aula02/ex6 test/ex6
cp -a aula02/ex6.out test/ex6.out
cp -a aula02/ex7 test/ex7
cp -a aula02/integer_arithmetic_pitfalls.c test/integer_arithmetic_pitfalls.c
cp -a aula02/primes.c test/primes.c
~/SO/SO_proj1/source$ ls ../test
~/SO/SO_proj1/source$
```

Figura 18: Teste da opção `-c`

```
~/SO/SO_proj1/source$ ls ../test
AED_Guiao_02.pdf  binary_search.c  ex1  ex2  ex2.c  ex3  ex3.c  ex4  ex4.c  ex5  ex5.c
      ex6  ex6.out  ex7  integer_arithmetic_pitfalls.c  primes.c
~/SO/SO_proj1/source$ rm -f ../test/*.c
~/SO/SO_proj1/source$ ./backup_files.sh ~/AED/aula02 ../test
cp -a aula02/binary_search.c test/binary_search.c
cp -a aula02/ex2.c test/ex2.c
cp -a aula02/ex3.c test/ex3.c
cp -a aula02/ex4.c test/ex4.c
cp -a aula02/ex5.c test/ex5.c
cp -a aula02/integer_arithmetic_pitfalls.c test/integer_arithmetic_pitfalls.c
cp -a aula02/primes.c test/primes.c
```

Figura 19: Teste de cópia parcial dos ficheiros

```
~/SO/SO_proj1/source$ ./backup_files.sh ~/AED/aula02 ../test
cp -a aula02/ex5.c test/ex5.c
```

Figura 20: Teste após se ter efetuado uma cópia para ../test e atualizado um ficheiro no diretório de trabalho

```
touch ../test/FicheiroNovo
~/SO/SO_proj1/source$ ./backup_files.sh ~/AED/aula02 ../test
~/SO/SO_proj1/source$ ls ../test/FicheiroNovo
ls: cannot access '../test/FicheiroNovo': No such file or directory
```

Figura 21: Teste para a eliminação de ficheiros que não fazem parte do diretório de trabalho

```
~/SO/SO_proj1/source$ ./backup_files.sh ~/AED/aula02 ../test
WARNING: backup entry test/ex2.c is newer than aula02/ex2.c; Should not happen
```

Figura 22: Teste para quando se atualiza um ficheiro no diretório de backup

```
~/SO/SO_proj1/source$ ./backup_files.sh -c ~/AED/aula02 ../test
ERROR: aula02/e.sh doesnt have writing permissions
cp -a aula02/AED_Guiao_02.pdf test/AED_Guiao_02.pdf
cp -a aula02/binary_search.c test/binary_search.c
cp -a aula02/ex1 test/ex1
cp -a aula02/ex2 test/ex2
cp -a aula02/ex2.c test/ex2.c
cp -a aula02/ex3 test/ex3
cp -a aula02/ex3.c test/ex3.c
cp -a aula02/ex4 test/ex4
cp -a aula02/ex4.c test/ex4.c
cp -a aula02/ex5 test/ex5
cp -a aula02/ex5.c test/ex5.c
cp -a aula02/ex6 test/ex6
cp -a aula02/ex6.out test/ex6.out
cp -a aula02/ex7 test/ex7
cp -a aula02/integer_arithmetic_pitfalls.c test/integer_arithmetic_pitfalls.c
cp -a aula02/primes.c test/primes.c
```

Figura 23: Teste para quando existe um ficheiro no diretório de backup a mais sem permissões de escrita

# 3 Segunda Iteração - backup.sh

O script `backup.sh`, semelhante ao script `backup_files.sh` copia os ficheiros da diretoria de trabalho, mas também copia diretorias, que podem ter sub-diretorias.

Este script, assim como o script `backup_files.sh` tem a opção "`checking`" que facilita a testagem do programa e também tem mais duas opções que permitem escolher se é feito ou não o backup de ficheiros ou diretorias específicas:

**Opção -c:** O script imprime tudo o que seria impresso se esta opção não fosse selecionada, mas não cria nem atualiza qualquer ficheiro, tal como no `backup_files.sh`.

**Opção -b:** O script recebe um ficheiro que contém uma lista de ficheiros e/ou diretorias que não devem ser copiados para a diretoria de backup.

**Opção -r:** O script irá apenas copiar os ficheiros que verificam a expressão regular(`REGEX`) recebida.

Para além destas opções este script assim como o script `backup_files.sh` recebe como argumentos o **Diretório de trabalho** e o **Diretório de backup**.

## 3.1 Inicializar as variáveis

Como o programa agora tem mais opção que a iteração anterior tivemos que acrescentar mais variáveis globais.

As variáveis `WORKDIR`, `BACKUP` e `CHECKING` mantêm-se igual à iteração anterior.

A variável `DIRS_FILE` indica o caminho absoluto ou relativo do ficheiro que contém a lista de ficheiros e/ou diretorias que não devem ser copiados para o diretório de backup caso seja selecionada a opção **-b**.

A variável `REGEX` indica a expressão regular que os ficheiros terão de verificar para ser copiados caso seja selecionada a opção **-r**.

A variável `DIRS` é um *array* que irá conter os caminhos absolutos ou relativos dos ficheiros e/ou diretórios que estão no ficheiro de texto com o caminho guardado na variável `DIRS_FILE`.

Também é declarado o `set DIRS_SET` que irá conter todos os caminhos que estarão no `array DIRS` para serem acessados com mais eficiência.

```
1 CHECKING="0"
2 DIRS_FILE=""
3 REGEX=""
4 DIRS=()
5 WORKDIR=""
6 BACKUP=""
7 declare -A DIRS_SET
```

Figura 24: Inicialização da variáveis globais

## 3.2 Verificação das Opções

Seguida da inicialização da variáveis o programa vai verificar quantas opções foram selecionadas. Se for adicionada uma opção não existente o programa imprime uma mensagem de erro e termina a sua execução. No final, são retirados os argumentos opcionais da lista de argumentos.

Se a opção **-c** for selecionada a variável `CHECKING` assume um valor igual **1**

Se a opção **-b** for selecionada a variável `DIRS_FILE` fica igual ao caminho do ficheiro que contém os caminhos dos ficheiros e/ou diretorias que não devem ser copiados. De seguida, verifica-se se o ficheiro existe e se é possível ler o ficheiro, se alguma destas condições não se verificar o programa dá uma mensagem de erro e termina a sua execução.

Se a opção **-r** for selecionada a variável `REGEX` fica igual à expressão regular dada. De seguida, é verificado se a expressão regular é valida, se não for o programa imprime uma mensagem de erro e termina a sua execução.

```
while getopts ":cb:r:" opt; do
    case $opt in
        c)
            CHECKING="1"
            ;;
        b)
            DIRS_FILE="$OPTARG"
            if [[ ! -f $DIRS_FILE || ! -r $DIRS_FILE ]]; then
                echo "$DIRS_FILE isn't a valid file"
                exit 1
            fi
            lines=()
            mapfile -t lines < "$DIRS_FILE"
            for line in "${lines[@]}"; do
                if [[ -e $(eval echo "$line") ]]; then
                    DIRS+=("$line")
                fi
            done
            ;;
        r)
            REGEX="$OPTARG"
            check_regex "$REGEX"
            if [[ $? -eq 1 ]]; then
                exit 1
            fi
            ;;
        \?)
            echo "ERROR: -$OPTARG is an invalid option"
            exit 1
            ;;
        :)
            echo "Option -$OPTARG requires an argument."
            exit 1
            ;;
    esac
done

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

Figura 25: Verificação das opções

A função `check_regex()` verifica se uma expressão regular é válida. Esta função funciona vendo se uma string de teste verifica a expressão regular dada. Se o valor de retornado dessa verificação for dois, isso significa que a expressão regular dada é inválida e o programa imprime uma mensagem de erro e termina a sua execução.

```
function check_regex() {
    local regex="$1"
    local test_string=""
    if [[ "$test_string" =~ $regex ]]; then
        :
    elif [[ $? -eq 2 ]]; then
        echo "ERROR: Invalid Regex"
        return 1
    fi
    return 0
}
```

Figura 26: Verificar se a expressão regular é válida

## 3.3 Validação dos argumentos

Como a maioria das verificações já tinha sido feita na primeira iteração, as validações mantiveram-se todas. O único problema novo que surgiu foi, a possibilidade ao colocarmos o segundo argumento como um diretório filho do primeiro, iria formar um *loop* infinito.

Para isso não acontecer, fizemos um *while loop* que irá verificar todos os diretórios que estão no caminho do backup, se algum deles for o diretório de trabalho, então o programa imprime uma mensagem de erro e termina a sua execução com o valor um.

```
while [[ "$BackupPath" != "/" ]]; do
    if [[ $WORKDIR == $BackupPath ]]; then
        echo "ERROR: "$(basename "$WORKDIR")" is parent of "$(basename "$BACKUP")""
        exit 1
    fi
    BackupPath="$(dirname "$BackupPath")"
done
```

Figura 27: Verificar se diretorio de trabalho é pai do diretorio de backup

## 3.4 Backup

Após todas as verificações o programa pode começar a fazer a cópia de segurança do diretório de trabalho, começado por adicionar todos os elementos do `array DIRS` ao `set DIRS_SET`.

```
for dir in "${DIRS[@]}"; do
    expanded_dir=$(eval echo "$dir")
    DIRS_SET["$(realpath "$expanded_dir")"]=1
done
```

Figura 28: Adicionar os elementos de DIRS a DIRS_SET

Depois, tornamos possível a cópia de ficheiros invisíveis(ficheiros começados por '.') e fazemos com que o programa não itere se não houver ficheiros em diretórios vazios.

```
shopt -s nullglob dotglob
```

Figura 29: Possibilita a cópia de ficheiros invisíveis

Terminando com a execução da função `backup()` que recebe como argumentos o diretório de trabalho e o diretório de backup, respetivamente.

```
backup "$WORKDIR" "$BACKUP"
```

Figura 30: Chamar a função backup

### 3.4.1 Função backup()

Esta função realiza cópia de segurança, aplicando as opções escolhidas. Esta função começa por chamar a função `backup_delete()` com o diretório de trabalho e o diretório de backup como argumentos. De seguida, começa a iterar por todos ficheiros e/ou diretórios do diretório de trabalho. Para cada elemento são feitas várias verificações:

Verificasse se o programa tem permissões para ler diretório de trabalho e se tem permissões para escrever no diretório de backup passado como argumento. Se alguma destas condições não se verificar o programa imprime uma mensagem de erro e retorna 1.

Chama a função `is_in_list()` para verificar se o elemento está lista de ficheiros/diretórios para ignorar, caso esteja este elemento vai ser ignorado.

Verifica se o elemento é um diretório. Se for, o programa vai executar a função `mkdirprint()` com os argumentos: nome do diretório caminho para o diretório de backup. Seguido de uma chamada recursiva da função `backup()` que recebe como argumentos o elemento e o diretório com o nome do elemento na diretoria de backup.

Se o elemento não for um diretório, então o programa vai ver se o ficheiro verifica a expressão regular guardada na variável `REGEX`. Se não verificar este vai ser ignorado, mas se verificar, vai ser executada a função `cpprint()` com o elemento e o nome do elemento no diretório de backup. Caso a opção **-r** não for selecionada, a variável `REGEX` contém uma expressão regular que é verifica por todas as strings.

```
1  function backup() {
2      backup_delete "$1" "$2"
3      if [ ! -r "$1" ]; then
4          echo "ERROR: "${1#$(dirname "$WORKDIR")/}" doenst have reading permissions"
5          return 1;
6      elif [[ -d "$2" ]] && [ ! -w "$2" ]; then
7          echo "ERROR: "${2#$(dirname "$BACKUP")/}" doenst have writing permissions"
8          return 1;
9      fi
10     for file in "$1"/*; do
11         if is_in_list "$file" "$DIRS_SET" ; then
12             continue;
13         fi
14         if [[ -d "$file" ]]; then
15             mkdirprint "$2/$(basename "$file")" "$BACKUP";
16             backup "$file" "$2/$(basename "$file")"
17             continue;
18         elif [[ ! "$(basename "$file")" =~ $REGEX ]]; then
19             continue;
20         fi
21         cpprint "$file" "$2/$(basename "$file")"
22     done
23 }
```

Figura 31: Função backup

### 3.4.2 Função backup_delete()

Esta função tem como objetivo apagar todos os ficheiros do diretório de backup que não existem no diretório de trabalho. A função começa por verificar se o diretório de backup existe, se não existir ele vai simplesmente retornar 0. Se ele existir, o programa começa a iterar por todos os elementos do diretório de backup e faz várias verificações:

Chama a função `is_in_list` e ignora o elemento caso a função retorne verdadeiro.

Verifica se tem permissões para escrever no elemento. Se não tiver o programa imprime uma mensagem de erro e ignora o elemento.

De segida, verifica se o elemento é um diretório. Se for, verifica se o elemento existe no diretório de trabalho, caso não exista e o checking(**-c**) não esteja ativado o elemento vai ser removido.

Se elemento não for um diretório o programa, se o elemento for um ficheiro, não estiver no diretório de trabalho e o checking(**-c**) não estiver ativo o elemento vai ser removido. Caso contrário ele vai ser ignorado

```
function backup_delete() {
    if [[ ! -d "$2" ]]; then
        return 0;
    fi
    for file in "$2"/*; do
        if is_in_list "$file" "$DIRS_SET" ; then
            continue;
        fi
        if [[ ! -w "$file" ]]; then
            echo "ERROR: "${file#$(dirname "$BACKUP")/}" doenst have permission to
    write"
            continue
        fi
        if [[ -d "$file" ]]; then
            if [[ ! -d "$1/$(basename "$file")" ]]; then
                if [[ $CHECKING -eq "0" ]]; then
                    rm -rf "$file"
                fi
            fi
            continue;
        fi
        if [[ ! -f "$file" || -f "$1/$(basename "$file")" ]]; then
            continue;
        fi
        if [[ $CHECKING -eq "0" ]]; then
            rm "$file"
        fi
    done
}
```

Figura 32: Função backup_delete()

### 3.4.3 Função is_in_list()

Esta função está no ficheiro `utils.sh` e tem como objetivo verificar se o ficheiro/diretório passado como argumento está na lista de ficheiros/diretórios que não devem ser copiados. Para isso, a função vai buscar o caminho absoluto do argumento passado e verifica se existe um valor associado a esse caminho no `set DIRS_SET`. Se o valor existir, ele vai retornar verdadeiro, caso contrario vai retornar falso.

```
function is_in_list(){
    local real_arg="$(realpath "$1")"
    [[ -n "${DIRS_SET[$real_arg]}" ]]
}
```

Figura 33: Função is_in_list()

### 3.5 Testes

Como o código de validação de variáveis é igual ao script `backup_files` se ocorrer um erro com os argumentos o output vai ser igual: backup_files testes.

```
~/Documents/24-25/1sem/so/SO_proj1/source $ ./backup.sh -c / /run/media/losg/
    EOS_202409/
ERROR: Not enough space in destination directory.
```

Figura 34: Teste quando diretório de backup não tem espaço suficiente

```
~/Documents/24-25/1sem/so/SO_proj1/source $ ./backup.sh -v . ../backup
ERROR: -v is an invalid option
```

Figura 35: Teste quando se usa uma flag não existente

```
~/Documents/24-25/1sem/so/SO_proj1/source $ ./backup.sh -b teste . ../backup
teste isn't a valid file
```

Figura 36: Teste quando se usa a flag -b com um ficheiro inválido

```
~/Documents/24-25/1sem/so/SO_proj1/source $ ./backup.sh -r [ . ../backup
ERROR: Invalid Regex
```

Figura 37: Teste quando se usa um regex inválido

Para estes teste usamos o seguinte diretório:

```
~/Documents/24-25/1sem/so/SO_proj1/source $ ls -A
 backup_check.sh   backup_summary.sh   dirs.txt         src           test_a1.tgz
 backup_files.sh   backup_test_a1      dontCopyMe.txt   test_a1.out   utils.sh
 backup.sh        'backup w.txt'       .file.txt        test_a1.sh
```

Figura 38: Diretório de teste

```
~/Documents/24-25/1sem/so/SO_proj1/source $ ls ../test
ls: cannot access '../test': No such file or directory
~/Documents/24-25/1sem/so/SO_proj1/source $ ./backup.sh . ../test
mkdir test
cp -a source/backup_check.sh test/backup_check.sh
cp -a source/backup_files.sh test/backup_files.sh
cp -a source/backup.sh test/backup.sh
cp -a source/backup_summary.sh test/backup_summary.sh
mkdir test/backup_test_a1
mkdir test/backup_test_a1/aaa
cp -a source/backup_test_a1/aaa/prog2.c test/backup_test_a1/aaa/prog2.c
cp -a source/backup_test_a1/output.txt test/backup_test_a1/output.txt
cp -a source/backup_test_a1/prog1.c test/backup_test_a1/prog1.c
cp -a source/backup w.txt test/backup w.txt
cp -a source/dirs.txt test/dirs.txt
cp -a source/dontCopyMe.txt test/dontCopyMe.txt
cp -a source/.file.txt test/.file.txt
mkdir test/src
mkdir test/src/aaa
cp -a source/src/aaa/prog2.c test/src/aaa/prog2.c
cp -a source/src/output.txt test/src/output.txt
cp -a source/src/prog1.c test/src/prog1.c
cp -a source/src/start.sh test/src/start.sh
cp -a source/test_a1.out test/test_a1.out
cp -a source/test_a1.sh test/test_a1.sh
cp -a source/test_a1.tgz test/test_a1.tgz
cp -a source/utils.sh test/utils.sh
~/Documents/24-25/1sem/so/SO_proj1/source $ ls -A ../test
 backup_check.sh   backup_summary.sh   dirs.txt         src           test_a1.tgz
 backup_files.sh   backup_test_a1      dontCopyMe.txt   test_a1.out   utils.sh
 backup.sh        'backup w.txt'       .file.txt        test_a1.sh
```

Figura 39: Teste sem opções

```
~/Documents/24-25/1sem/so/SO_proj1/source $ ls ../test
ls: cannot access '../test': No such file or directory
~/Documents/24-25/1sem/so/SO_proj1/source $ ./backup.sh -c . ../test
mkdir test
cp -a source/backup_check.sh test/backup_check.sh
cp -a source/backup_files.sh test/backup_files.sh
cp -a source/backup.sh test/backup.sh
cp -a source/backup_summary.sh test/backup_summary.sh
mkdir test/backup_test_a1
mkdir test/backup_test_a1/aaa
cp -a source/backup_test_a1/aaa/prog2.c test/backup_test_a1/aaa/prog2.c
cp -a source/backup_test_a1/output.txt test/backup_test_a1/output.txt
cp -a source/backup_test_a1/prog1.c test/backup_test_a1/prog1.c
cp -a source/backup w.txt test/backup w.txt
cp -a source/dirs.txt test/dirs.txt
cp -a source/dontCopyMe.txt test/dontCopyMe.txt
cp -a source/.file.txt test/.file.txt
mkdir test/src
mkdir test/src/aaa
cp -a source/src/aaa/prog2.c test/src/aaa/prog2.c
cp -a source/src/output.txt test/src/output.txt
cp -a source/src/prog1.c test/src/prog1.c
cp -a source/src/start.sh test/src/start.sh
cp -a source/test_a1.out test/test_a1.out
cp -a source/test_a1.sh test/test_a1.sh
cp -a source/test_a1.tgz test/test_a1.tgz
cp -a source/utils.sh test/utils.sh
~/Documents/24-25/1sem/so/SO_proj1/source $ ls ../test
ls: cannot access '../test': No such file or directory
```

Figura 40: Teste com opção -c

```
~/Documents/24-25/1sem/so/SO_proj1/source $ ls ../test
~/Documents/24-25/1sem/so/SO_proj1/source $ cat dirs.txt
/home/losg/Documents/c/
backup.sh
src/
/home/losg/Documents/24-25/1sem/so/SO_proj1/source/backup_test_a1/
~/SO/projeto1/back/balls.txt
chaliça/
~/AED/aula04/
~/SO/SO_proj1/source/backup w.txt
~/Documents/24-25/1sem/so/SO_proj1/source $ ./backup.sh -b dirs.txt . ../test
cp -a source/backup_check.sh test/backup_check.sh
cp -a source/backup_files.sh test/backup_files.sh
cp -a source/backup_summary.sh test/backup_summary.sh
cp -a source/backup w.txt test/backup w.txt
cp -a source/dirs.txt test/dirs.txt
cp -a source/dontCopyMe.txt test/dontCopyMe.txt
cp -a source/test_a1.out test/test_a1.out
cp -a source/test_a1.sh test/test_a1.sh
cp -a source/test_a1.tgz test/test_a1.tgz
cp -a source/utils.sh test/utils.sh
~/Documents/24-25/1sem/so/SO_proj1/source $ ls -A ../test
 backup_check.sh   backup_files.sh   backup_summary.sh  'backup w.txt'   dirs.txt
    dontCopyMe.txt   .file.txt   test_a1.out   test_a1.sh   test_a1.tgz   utils.
    sDocuments/24-25/1sem/so/SO_proj1/source $ ls hDocuments/24-25/1sem/so/SO_proj1/
    source $ ls
```

Figura 41: Teste com a opção -b

```
~/Documents/24-25/1sem/so/SO_proj1/source $ ls ../test/
~/Documents/24-25/1sem/so/SO_proj1/source $ ./backup.sh -r ^p . ../test/
mkdir test/backup_test_a1
mkdir test/backup_test_a1/aaa
cp -a source/backup_test_a1/aaa/prog2.c test/backup_test_a1/aaa/prog2.c
cp -a source/backup_test_a1/prog1.c test/backup_test_a1/prog1.c
mkdir test/src
mkdir test/src/aaa
cp -a source/src/aaa/prog2.c test/src/aaa/prog2.c
cp -a source/src/prog1.c test/src/prog1.c
~/Documents/24-25/1sem/so/SO_proj1/source $ ls ../test/
backup_test_a1  src
~/Documents/24-25/1sem/so/SO_proj1/source $ ls ../test/backup_test_a1/
aaa  prog1.c
~/Documents/24-25/1sem/so/SO_proj1/source $ ls ../test/backup_test_a1/aaa/
prog2.c
```

Figura 42: Teste com a opção -r

```
~/Documents/24-25/1sem/so/SO_proj1/source $ ls -A ../test/
~/Documents/24-25/1sem/so/SO_proj1/source $ cat dirs.txt
/home/losg/Documents/c/
backup.sh
src/
/home/losg/Documents/24-25/1sem/so/SO_proj1/source/backup_test_a1/
~/SO/projeto1/back/balls.txt
chaliça/
~/AED/aula04/
~/SO/SO_proj1/source/backup w.txt
~/Documents/24-25/1sem/so/SO_proj1/source $ ./backup.sh -b dirs.txt -r ^b . ../test
cp -a source/backup_check.sh test/backup_check.sh
cp -a source/backup_files.sh test/backup_files.sh
cp -a source/backup_summary.sh test/backup_summary.sh
cp -a source/backup w.txt test/backup w.txt
~/Documents/24-25/1sem/so/SO_proj1/source $ ls -A ../test/
 backup_check.sh   backup_files.sh   backup_summary.sh  'backup w.txt'
```

Figura 43: Teste com a opção -b e -r

```
~/Documents/24-25/1sem/so/SO_proj1/source $ ls -A ../test/
~/Documents/24-25/1sem/so/SO_proj1/source $ cat dirs.txt
/home/losg/Documents/c/
backup.sh
src/
/home/losg/Documents/24-25/1sem/so/SO_proj1/source/backup_test_a1/
~/SO/projeto1/back/balls.txt
chaliça/
~/AED/aula04/
~/SO/SO_proj1/source/backup w.txt
~/Documents/24-25/1sem/so/SO_proj1/source $ ./backup.sh -c -b dirs.txt -r ^b . ../test
cp -a source/backup_check.sh test/backup_check.sh
cp -a source/backup_files.sh test/backup_files.sh
cp -a source/backup_summary.sh test/backup_summary.sh
cp -a source/backup w.txt test/backup w.txt
~/Documents/24-25/1sem/so/SO_proj1/source $ ls -A ../test/
~/Documents/24-25/1sem/so/SO_proj1/source $
```

Figura 44: Teste com todas as opções

# 4 Terceira Iteração - backup_summary.sh

Este script é a última iteração dos programas que têm como objetivo fazer backup, este script expande em complexidade às versões anteriores, porque no final de se fazer a cópia de cada diretoria ele imprime um sumário que indica o número de erros, de avisos, de ficheiros atualizados, de ficheiros copiados e de ficheiros eliminados. Para além disso, também aparece o tamanho em bytes do total dos ficheiros copiados e o mesmo para os eliminados. Apesar de parecer uma alteração simples, nós tivemos que reformular várias partes do código para o programa executar como desejado.

## 4.1 Inicialização das variáveis globais

Esta parte do programa, manteve-se igual à versão apresentada no `backup.sh`, apesar de termos acrescentado uma nova variável, chamada de `ARG_ERRORS` que tem como objetivo medir o número de erros presentes nos argumentos, assim quando for feito o sumário da execução, o número de erros exibidos é o correto. Esta variável começa com o valor zero.

## 4.2 Verificação das Opções

Esta parte do programa também é bastante semelhante à versão apresentada no `backup.sh`, mas como agora queremos imprimir o sumário, em vez de terminarmos a execução, incrementamos a variável `ARG_ERRORS` e testamos os restantes argumentos opcionais.

```
1    \?)
2        echo "ERROR: -$OPTARG is an invalid option"
3        exit 1
4        ;;
```

Figura 45: Exemplo de como se lida com um erro dos argumentos opcionais no `backup.sh`

```
1    \?)
2        echo "ERROR: -$OPTARG is an invalid option"
3        ((ARG_ERRORS++))
4        ;;
```

Figura 46: Exemplo da mudança feita no `backup_summary.sh`

## 4.3 Validação dos argumentos

Novamente grande parte do código das verificações, provém do `backup.sh`. Na verificação do número de argumentos, também mudamos para apenas aumentar a variável `ARG_ERRORS`, mas após esta se houver erros, a execução será terminada depois da função `summary()` ter sido chamada. Tirando os dois primeiros argumentos, os outros serão zeros, porque o backup ainda não começou a ser feito, então não foram copiados nem eliminados ficheiro.

```
1    if [[ ! $ARG_ERRORS -eq 0 ]]; then
2        summary "$WORKDIR" "$ARG_ERRORS" "0" "0" "0" "0" "0" "0"
3        exit 1
4    fi
```

Figura 47: Verificação da existência de erros nos argumentos opcionais

A seguir desta verificação, as próximas implicam que o programa termine a sua execução caso os argumentos tenham algum problema, isto é porque a partir de agora, as verificações envolvem verificar os diretórios em si. Assim, a única mudança que foi feita nestas verificações foi que antes de terminar a execução, o programa chama a função `summary()`, onde o primeiro argumento é o diretório de trabalho, o segundo é um, visto que, até lá não foi detetado mais nenhum erro, e os restantes serão zero.

## 4.4 Função summary()

Como o script implica que seja impresso várias vezes o sumário de cada backup, achámos mais simples criar uma função que fizesse isso. Esta tem 8 argumentos:

**$1**: Argumento que especifica o caminho absoluto do diretório que acabou de ser copiado.

**$2**: Número de erros obtidos durante o backup.

**$2**: Número de erros obtidos durante o backup.

**$3**: Número de avisos impressos durante o backup.

**$4**: Número de ficheiros atualizados durante o backup.

**$5**: Número de ficheiros copiados durante o backup.

**$6**: Tamanho total de todos os ficheiros copiados no backup.

**$7**: Número de ficheiros eliminados durante o backup.

**$8**: Tamanho total de todos os ficheiros eliminados no backup.

A função é bastante simples, esta apenas obtém o caminho simplificado do diretório que foi copiado e depois imprime o sumário conforme os argumentos fornecidos.

```
function summary() {
    local simpler_name="${1#$(dirname "$WORKDIR")/}"
    echo "While backuping "$simpler_name": $2 Errors; $3 Warnings; $4 Updated; $5
    Copied (${6}B); $7 Deleted (${8}B)"
}
```

Figura 48: Função `summary()`

## 4.5 Função backup()

Esta função é a principal mudança desta iteração. Em primeiro lugar optamos por juntar as funções `backup()` e `backup_delete()`. Decidimos fazer isso porque, para imprimir o sumário, é necessário vários dados sobre as operações que estas duas funções fazem e como alguns são números que podem ser facilmente maiores que 255 não é viável estar a retornar esses valores da função e depois fazer o sumário. Também pensamos em adicionar novas variáveis globais, mas como função é recursiva também não era muito prático fazer dessa forma. Desta forma, executámos todas as operações na mesma função, por questões de praticidade. A função continua a manter os mesmos argumentos de entrada.

### 4.5.1 Inicialização das variáveis locais

Tal como tinha sido mencionado, precisamos de várias variáveis para depois imprimir o sumário. Por isso, a função começa com a inicialização destas variáveis como zero e ao longo do programa estas poderão ser incrementadas. Estas variáveis são todas locais para não haver problemas no caso de haver recursão.

```
    # Variáveis para a função summary()
    local ERRORS="0"
    local WARNINGS="0"
    local FILES_UPDATED="0"
    local FILES_COPIED="0"
    local FILES_DELETED="0"
    local SIZE_COPIED="0"
    local SIZE_REMOVED="0"
}
```

Figura 49: Variáveis locais da função `backup()`

### 4.5.2 Verificação dos argumentos

Como esta função é recursiva, então é necessário que os argumentos sejam sempre verificados, então a função, após definir as variáveis locais, verifica se o diretório de trabalho tem permissão de leitura e se o diretório de backup tem permissão de escrita. Se faltar alguma destas permissões, o processo não pode continuar, então a função imprime uma mensagem de erro e o sumário, para depois retornar um.

```
1     if [ ! -r "$1" ]; then
2         echo "ERROR: "${1#$(dirname "$WORKDIR")/}" doenst have reading permissions"
3         summary "$1" "$ERRORS" "$WARNINGS" "$FILES_UPDATED" "$FILES_COPIED" "
    $SIZE_COPIED" "$FILES_DELETED" "$SIZE_REMOVED"
4         ((ERRORS++))
5         return 1;
6     elif [ ! -w "$2" ]; then
7         echo "ERROR: "${2#$(dirname "$BACKUP")/}" doenst have writing permissions"
8         summary "$1" "$ERRORS" "$WARNINGS" "$FILES_UPDATED" "$FILES_COPIED" "
    $SIZE_COPIED" "$FILES_DELETED" "$SIZE_REMOVED"
9         ((ERRORS++))
10        return 1;
11    fi
```

Figura 50: Verificação dos argumentos

### 4.5.3 Realização da Cópia de Segurança

Esta parte do código era a função `backup()` na iteração anterior, novamente tentamos reutilizar o máximo de código que conseguimos, assim esta parte está praticamente igual à iteração anterior mas agora após usarmos a função `cpprint()` verificamos qual será o seu valor de retorno e mediante o resultado, incrementamos as variáveis adequadas.

```
1     for file in "$1"/*; do
2         if is_in_list "$file" "$DIRS_SET" ; then
3             continue;
4         fi
5         if [[ -d "$file" ]]; then
6             mkdirprint "$2/$(basename "$file")" "$BACKUP";
7             backup "$file" "$2/$(basename "$file")"
8             continue;
9         elif [[ ! "$(basename "$file")" =~ $REGEX ]]; then
10            continue;
11        fi
12        local file_copy="$2/$(basename "$file")"
13        cpprint "$file" "$file_copy"
14        local ret=$?
15        if [[ $ret -eq  0 ]]; then
16            ((FILES_COPIED++))
17            ((SIZE_COPIED+=$(stat -c %s "$file")))
18        elif [[ $ret -eq  1 ]]; then
19            ((FILES_UPDATED++))
20        elif [[ $ret -eq  3 ]]; then
21            ((WARNINGS++))
22        elif [[ $ret -eq  4 ]]; then
23            ((ERRORS++))
24        fi
25    done
```

Figura 51: Componente da função `backup()` que realiza a cópia de segurança

### 4.5.4 Remoção dos Ficheiros a Mais

Após fazermos a cópia de segurança ainda podem haver ficheiros a mais que já estavam antes do programa ser executado, por essa razão é necessário remover estes ficheiros. Para o fazer, adaptámos o código da função `backup_delete()`. A única coisa que foi alterada foi que agora, antes de remover alguma coisa, somamos, às variáveis locais adequadas, o espaço que cada ocupa e o número de ficheiros que estas têm.

```
1     if [[ ! -d "$2" ]]; then
2         summary "$1" "$ERRORS" "$WARNINGS" "$FILES_UPDATED" "$FILES_COPIED" "
      $SIZE_COPIED" "$FILES_DELETED" "$SIZE_REMOVED"
3         return 0;
4     fi
5
6     for file in "$2"/*; do
7         if is_in_list "$file" "$DIRS_SET" ; then
8             continue;
9         fi
10        if [ ! -w "$file" ]; then
11            echo "ERROR: "${file#$(dirname "$BACKUP")/}" doenst have writing
      permissions"
12            ((ERRORS++))
13            continue;
14        fi
15        if [[ -d "$file" ]]; then
16            if [[ ! -d "$1/$(basename "$file")" ]]; then
17                local directory_size=$(( $(du -sk "$file" | awk '{print $1}') * 1024 )
      )
18                ((SIZE_REMOVED+=$directory_size))
19                local file_count=$(find "$file" -type f | wc -l)
20                ((FILES_DELETED+=$file_count))
21                if [[ $CHECKING -eq "0" ]]; then
22                    rm -rf "$file"
23                fi
24            fi
25            continue;
26        fi
27        if [[ -f "$1/$(basename "$file")" ]]; then
28            continue;
29        fi
30        ((SIZE_REMOVED+=$(stat -c %s "$file") ))
31        ((FILES_DELETED++))
32        if [[ $CHECKING -eq "0" ]]; then
33            rm "$file"
34        fi
35    done
```

Figura 52: Componente da função `backup()` que elimina os ficheiros que não fazem parte da diretório de trabalho

### 4.5.5 Impressão do sumário

Agora que as operações necessárias foram feitas, só falta imprimir o sumário com os dados pedidos no enunciado. Para o fazer, basta evocar a função `summary()` com o primeiro argumento como o primeiro argumento desta função e os restantes serão as variáveis locais.

```
1     summary "$1" "$ERRORS" "$WARNINGS" "$FILES_UPDATED" "$FILES_COPIED" "$SIZE_COPIED"
       "$FILES_DELETED" "$SIZE_REMOVED"
```

Figura 53: Evocação da função `summary()`

## 4.6 Testes

Novamente, tiveram de ser feitos mais testes para assegurar que o novo código está a correr bem e não entra em conflito com o código das iterações anteriores.

```
~/SO/SO_proj1/source$ ./backup_summary.sh -c -r "(a" ../naoexitiso .
ERROR: Invalid Regex
ERROR: naoexitiso is not a directory
While backuping ../naoexitiso: 2 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0
    Deleted (0B)
```

Figura 54: Teste de vários erros nos argumentos

```
~/SO/SO_proj1/source$ ./backup_summary.sh .. .
ERROR: SO_proj1 is parent of source
While backuping SO_proj1: 1 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0
    B)
```

Figura 55: Teste de quando o primeiro argumento é pai do segundo

Para realizar os testes seguintes foram usados o seguinte diretório:

```
~/SO/SO_proj1/source$ ls -R .
.:
 backup_check.sh   backup.sh           backup_test_a1   dirs.txt         rt    test_a1
    .out   test_a1.tgz
 backup_files.sh   backup_summary.sh  'backup w.txt'    dontCopyMe.txt   src   test_a1
    .sh    utils.sh

./backup_test_a1:
aaa  output.txt  prog1.c

./backup_test_a1/aaa:
prog2.c

./src:
aaa  output.txt  prog1.c  start.sh

./src/aaa:
prog2.c
```

Figura 56: Diretório usado para os testes seguintes

```
~/SO/SO_proj1/source$ ./backup_summary.sh . ../test
mkdir test
cp -a source/backup_check.sh test/backup_check.sh
cp -a source/backup_files.sh test/backup_files.sh
cp -a source/backup.sh test/backup.sh
cp -a source/backup_summary.sh test/backup_summary.sh
mkdir test/backup_test_a1
mkdir test/backup_test_a1/aaa
cp -a source/backup_test_a1/aaa/prog2.c test/backup_test_a1/aaa/prog2.c
While backuping source/backup_test_a1/aaa: 0 Errors; 0 Warnings; 0 Updated; 1 Copied
    (8282B); 0 Deleted (0B)
cp -a source/backup_test_a1/output.txt test/backup_test_a1/output.txt
cp -a source/backup_test_a1/prog1.c test/backup_test_a1/prog1.c
While backuping source/backup_test_a1: 0 Errors; 0 Warnings; 0 Updated; 2 Copied (8292
    B); 0 Deleted (0B)
cp -a source/backup w.txt test/backup w.txt
cp -a source/dirs.txt test/dirs.txt
cp -a source/dontCopyMe.txt test/dontCopyMe.txt
cp -a source/.file.txt test/.file.txt
mkdir test/src
mkdir test/src/aaa
cp -a source/src/aaa/prog2.c test/src/aaa/prog2.c
While backuping source/src/aaa: 0 Errors; 0 Warnings; 0 Updated; 1 Copied (8282B); 0
    Deleted (0B)
cp -a source/src/output.txt test/src/output.txt
cp -a source/src/prog1.c test/src/prog1.c
cp -a source/src/start.sh test/src/start.sh
While backuping source/src: 0 Errors; 0 Warnings; 0 Updated; 3 Copied (16596B); 0
    Deleted (0B)
cp -a source/test_a1.out test/test_a1.out
cp -a source/test_a1.sh test/test_a1.sh
cp -a source/test_a1.tgz test/test_a1.tgz
cp -a source/utils.sh test/utils.sh
While backuping source: 0 Errors; 0 Warnings; 0 Updated; 12 Copied (19355B); 0 Deleted
     (0B)
```

Figura 57: Teste de cópia para um diretório inexistente

```
~/SO/SO_proj1/source$ cat dirs.txt
/home/losg/Documents/c/
backup.sh
~/AED/aula04/
~/SO/SO_proj1/source/backup w.txt
~/SO/SO_proj1/source$ ./backup_summary.sh -c -b dirs.txt -r b . ../test
mkdir test
cp -a source/backup_check.sh test/backup_check.sh
cp -a source/backup_files.sh test/backup_files.sh
cp -a source/backup_summary.sh test/backup_summary.sh
mkdir test/backup_test_a1
mkdir test/backup_test_a1/aaa
While backuping source/backup_test_a1/aaa: 0 Errors; 0 Warnings; 0 Updated; 0 Copied
    (0B); 0 Deleted (0B)
While backuping source/backup_test_a1: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B);
     0 Deleted (0B)
mkdir test/src
mkdir test/src/aaa
While backuping source/src/aaa: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0
    Deleted (0B)
While backuping source/src: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted
    (0B)
While backuping source: 0 Errors; 0 Warnings; 0 Updated; 3 Copied (9845B); 0 Deleted
    (0B)
~/SO/SO_proj1/source$ ls ../test
ls: cannot access '../test': No such file or directory
```

Figura 58: Teste com mútiplas flags

```
~/SO/SO_proj1/source$ cat dirs.txt
/home/losg/Documents/c/
backup.sh
~/SO/SO_proj1/source/src/
~/AED/aula04/
~/SO/SO_proj1/source/backup w.txt
~/SO/SO_proj1/source$ ./backup_summary.sh -c -b dirs.txt . ../test
cp -a source/backup_check.sh test/backup_check.sh
cp -a source/backup_files.sh test/backup_files.sh
cp -a source/backup_summary.sh test/backup_summary.sh
mkdir test/backup_test_a1
mkdir test/backup_test_a1/aaa
cp -a source/backup_test_a1/aaa/prog2.c test/backup_test_a1/aaa/prog2.c
While backuping source/backup_test_a1/aaa: 0 Errors; 0 Warnings; 0 Updated; 1 Copied
    (8282B); 0 Deleted (0B)
cp -a source/backup_test_a1/output.txt test/backup_test_a1/output.txt
cp -a source/backup_test_a1/prog1.c test/backup_test_a1/prog1.c
While backuping source/backup_test_a1: 0 Errors; 0 Warnings; 0 Updated; 2 Copied (8292
    B); 0 Deleted (0B)
cp -a source/dirs.txt test/dirs.txt
cp -a source/dontCopyMe.txt test/dontCopyMe.txt
cp -a source/.file.txt test/.file.txt
cp -a source/test_a1.out test/test_a1.out
cp -a source/test_a1.sh test/test_a1.sh
cp -a source/test_a1.tgz test/test_a1.tgz
cp -a source/utils.sh test/utils.sh
While backuping source: 0 Errors; 0 Warnings; 0 Updated; 10 Copied (15241B); 1 Deleted
     (16016B)
```

Figura 59: Teste com um ficheiro no diretório de backup

```
~/SO/SO_proj1/source$ ./backup_summary.sh . ../test36
While backuping source/backup_test_a1/aaa: 0 Errors; 0 Warnings; 0 Updated; 0 Copied
    (0B); 0 Deleted (0B)
While backuping source/backup_test_a1: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B);
     0 Deleted (0B)
WARNING: backup entry test/dirs.txt is newer than source/dirs.txt; Should not happen
While backuping source/src/aaa: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0
    Deleted (0B)
While backuping source/src: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted
    (0B)
WARNING: backup entry test/test_a1.out is newer than source/test_a1.out; Should not
    happen
While backuping source: 0 Errors; 2 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
```

Figura 60: Teste com ficheiros atualizados no diretório de backup

```
~/SO/SO_proj1/source$ chmod -r ./src
~/SO/SO_proj1/source$ chmod -r ./dontCopyMe.txt
~/SO/SO_proj1/source$ ./backup_summary.sh . ../test
cp -a source/backup_check.sh test/backup_check.sh
cp -a source/backup_files.sh test/backup_files.sh
cp -a source/backup.sh test/backup.sh
cp -a source/backup_summary.sh test/backup_summary.sh
mkdir test/backup_test_a1
mkdir test/backup_test_a1/aaa
cp -a source/backup_test_a1/aaa/prog2.c test/backup_test_a1/aaa/prog2.c
While backuping source/backup_test_a1/aaa: 0 Errors; 0 Warnings; 0 Updated; 1 Copied
    (8282B); 0 Deleted (0B)
cp -a source/backup_test_a1/output.txt test/backup_test_a1/output.txt
cp -a source/backup_test_a1/prog1.c test/backup_test_a1/prog1.c
While backuping source/backup_test_a1: 0 Errors; 0 Warnings; 0 Updated; 2 Copied (8292
    B); 0 Deleted (0B)
cp -a source/backup w.txt test/backup w.txt
cp -a source/dirs.txt test/dirs.txt
ERROR: source/dontCopyMe.txt doenst have permission to read
cp -a source/.file.txt test/.file.txt
mkdir test/src
ERROR: source/src doenst have reading permissions
While backuping source/src: 1 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted
    (0B)
cp -a source/test_a1.out test/test_a1.out
cp -a source/test_a1.sh test/test_a1.sh
cp -a source/test_a1.tgz test/test_a1.tgz
cp -a source/utils.sh test/utils.sh
While backuping source: 1 Errors; 0 Warnings; 0 Updated; 11 Copied (19328B); 1 Deleted
     (16016B)
```

Figura 61: Teste com um ficheiros e diretórios sem permissões de leitura

```
~/SO/SO_proj1/source$ chmod -w ../test/.file.txt
~/SO/SO_proj1/source$ chmod -w ../test/backup_test_a1
~/SO/SO_proj1/source$ ./backup_summary.sh . ../test
cp -a source/backup.sh test/backup.sh
cp -a source/backup_summary.sh test/backup_summary.sh
ERROR: test/backup_test_a1 doenst have writing permissions
While backuping source/backup_test_a1: 1 Errors; 0 Warnings; 0 Updated; 0 Copied (0B);
     0 Deleted (0B)
ERROR: test/.file.txt doenst have permission to write
While backuping source/src/aaa: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0
    Deleted (0B)
While backuping source/src: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted
    (0B)
While backuping source: 1 Errors; 0 Warnings; 2 Updated; 0 Copied (0B); 0 Deleted (0B)
```

Figura 62: Teste com atualização de ficheiros e diretórios sem permissões de escrita

# 5 Função backup_check.sh

Este script, ao contrário dos outros, não vai fazer uma cópia de segurança do diretório de trabalho. Este vai apenas verificar se o conteúdo dos ficheiros da diretoria de backup são iguais ao conteúdo dos ficheiros correspondentes na diretoria de trabalho. Imprimindo as diferenças sempre que as encontrar.

## 5.1 Verificação dos Argumentos

O programa começa por verificar se o número de argumentos é igual a dois, e se os argumentos são diretórios. Se alguma das condições não se verificar, o programa imprime uma mensagem de erro e termina a sua execução.

```
if [[ ! $# -eq 2 ]]; then
    echo "ERROR: The function has two arguments"
    exit 1
elif [[ ! -d "$1" ]]; then
    echo "ERROR: "$(basename "$1")" is not a directory"
    exit 1;
elif [[ ! -d "$2" ]]; then
    echo "ERROR: "$(basename "$2")" is not a directory"
    exit 1;
fi
```

Figura 63: Verificar argumentos

Depois de verificar os argumentos, estes são guardados em variáveis, o diretório de trabalho na variável `WORKDIR` e o diretório de backup na variável `BACKUP`. De seguida, verifica se tem permissão para ler ambos os diretórios, dando erro se não tiver.

```
WORKDIR="$(realpath "$1")"
BACKUP="$(realpath "$2")"

if [[ ! -r "$1" ]]; then
    echo "ERROR: "$(basename "$WORKDIR")" doesnt have read permissions"
    exit 1
elif [[ ! -r "$2" ]]; then
    echo "ERROR: "$(basename "$BACKUP")" doesnt have read permissions"
    exit 1
fi
```

Figura 64: Verificar permissões

## 5.2 Backup Check

Depois das verificações o programa pode começar a verificar se o conteúdo dos ficheiros da diretoria de backup é igual ao conteúdo dos ficheiros correspondentes na diretoria de trabalho.

O programa começa por permitir a leitura dos ficheiros invisíveis(começados por '.'), seguido da chamada da função `check_content()` recebendo como argumentos o diretório de trabalho e o diretório de backup.

```
shopt -s nullglob dotglob

check_content "$WORKDIR" "$BACKUP"
```

Figura 65: Começar a verificar o conteúdo dos ficheiros

### 5.2.1 Função check_content

Esta função tem como objetivo verificar se o conteúdo de dois ficheiros é igual e se não for, esta imprime uma mensagem de erro. Esta começa por iterar sob todos os elementos do diretório de trabalho e fazendo várias verificações.

Verifica se tem permissão para ler o elemento, dando uma mensagem de erro e ignorando-o se não a tiver.

Depois, verifica se o elemento é um diretório, se for o programa vai fazer uma chamada recursiva da função `check_content`.

Se for um ficheiro, verifica se o ficheiro existe no diretório de backup, se não existir é ignorado. Se ele existir, verifica se tem permissões de leitura. Caso não tenha, é impressa uma mensagem de erro e este é ignorado.

Se o ficheiro passar as verificações anteriores, o programa vai usar o comando `md5sum` para ir buscar a *hash* do ficheiro do diretório de trabalho e do diretório de backup. Se as *hashes* forem diferentes, significa que o conteúdo é diferente, logo o programa imprime uma mensagem de erro.

```
1  function check_content() {
2      for file in "$1"/*; do
3          local basename=$(basename "$file")
4          local simpler_name_workdir="${1#$(dirname "$WORKDIR")/}"
5          if [[ ! -r "$file" ]]; then
6              echo "ERROR: "$simpler_name_workdir"/"$basename" doesnt have read
       permissions"
7              continue;
8          fi
9          if [[ -d "$file" ]]; then
10             check_content "$file" "$2/$(basename  "$file")"
11         fi
12         if [[ ! -f "$2"/"$basename" ]];then
13             continue
14         fi
15         local simpler_name_backup="${2#$(dirname "$BACKUP")/}"
16         if [[ ! -r "$2"/"$basename" ]]; then
17             echo "ERROR: "$simpler_name_backup"/"$basename" doesnt have read
       permissions"
18             continue;
19         fi
20         local original_hash=$(md5sum "$file" | awk '{ print $1 }')
21         local backup_hash=$(md5sum "$2/$basename" | awk '{ print $1 }')
22         if [[ "$original_hash" != "$backup_hash" ]]; then
23             echo ""$simpler_name_workdir"/"$basename" "$simpler_name_backup"/"
       $basename" differ"
24         fi
25     done
26 }
```

Figura 66: Função check_content

## 5.3 Testes

```
~/Documents/24-25/1sem/so/SO_proj1/source $ ./backup_check.sh . ../test/
~/Documents/24-25/1sem/so/SO_proj1/source $
```

Figura 67: O conteúdo do diretório de backup não difere do conteúdo do diretório de trabalho

```
~/Documents/24-25/1sem/so/SO_proj1/source $ ./backup_check.sh . ../test/
~/Documents/24-25/1sem/so/SO_proj1/source $ echo teste > dirs.txt
~/Documents/24-25/1sem/so/SO_proj1/source $ ./backup_check.sh . ../test/
source/dirs.txt test/dirs.txt differ
~/Documents/24-25/1sem/so/SO_proj1/source $
```

Figura 68: O conteúdo do diretório de backup difere do conteúdo do diretório de trabalho

```
~/Documents/24-25/1sem/so/SO_proj1/source $ ./backup_check.sh . ../test/
~/Documents/24-25/1sem/so/SO_proj1/source $ echo "teste" > backup_test_a1/aaa/prog2.c
~/Documents/24-25/1sem/so/SO_proj1/source $ ./backup_check.sh . ../test/
source/backup_test_a1/aaa/prog2.c test/backup_test_a1/aaa/prog2.c differ
~/Documents/24-25/1sem/so/SO_proj1/source $
```

Figura 69: O conteúdo de um ficheiro de um subdiretório difere do seu ficheiro correspondente no diretório de backup

```
~/SO/SO_proj1/source$ chmod -r ../test/ex7
~/SO/SO_proj1/source$ ./backup_check.sh ~/AED/aula02 ../test
ERROR: test/ex7 doesnt have read permissions
```

Figura 70: Teste de ficheiros sem permissão de leitura

```
~/SO/SO_proj1/source$ ./backup_check.sh ~/AED/aula02 ../tes
ERROR: tes is not a directory
```

Figura 71: Teste de argumentos que não são diretórios

```
~/SO/SO_proj1/source$ bash ./backup_check.sh ~/AED/aula02
ERROR: The function has two arguments
```

Figura 72: Teste com apenas dois argumentos

# 6 Conclusão

Este trabalho permitiu explorar e consolidar as nossas habilidades em Bash através do desenvolvimento de scripts para a realização de backups. Com esta abordagem iterativa, foi possível alcançar o objetivo final, permitindo-nos compreender a importância de, primeiro, simplificar os problemas para, depois, desenvolver o produto final.

Percebemos também a importância de realizar testes, visto que descobrimos vários erros com eles. E sem estes não podíamos garantir a segurança do script em diferentes cenários. Com este trabalho ganhámos noção das possibilidades do uso do Bash para automatizar tarefas no ambiente de Linux, sendo uma solução poderosa e flexível na gestão de sistemas operativos.

# 7 Webgrapfia


**GNU Bash Reference Manual**
Disponível em: https://www.gnu.org/software/bash/manual/ Fonte oficial para comandos e funcionalidades do Bash.

**Linux Man Pages**
Disponível em: https://man7.org/linux/man-pages/ Recurso essencial para compreender comandos como `cp`, `mkdir`, `rm` e `md5sum`.

**Advanced Bash-Scripting Guide**
Disponível em: https://tldp.org/LDP/abs/html/ Guia abrangente para scripting avançado em Bash.

**Stack Overflow**
Disponível em: https://stackoverflow.com/ Fórum com soluções práticas e discussões sobre problemas comuns em Bash.

**Regular-Expressions.info**
Disponível em: https://www.regular-expressions.info/ Referência útil para entender expressões regulares usadas no script.