Departamento de Eletrónica, Telecomunicações e Informática

Ferramenta de criação/atualização de cópias de segurança em Bash

Sistemas Operativos

Diogo Teixeira - 113042

Prof.: Nuno Lau (nunolau@ua.pt)

17 de novembro de 2024

Índice

Capítulo 1

Introdução

Este projeto tem como objetivo o desenvolvimento de um conjunto de *scripts* em *Bash* que permitem automatizar a criação e atualização de cópias de segurança (*backups*) de uma diretoria de trabalho para uma diretoria de *backup*. A solução foi projetada para ser eficiente, realizando cópias incrementais em execuções subsequentes, ou seja, copiando apenas os ficheiros novos ou modificados e removendo aqueles que deixaram de existir na diretoria de origem. Esta abordagem otimiza o tempo de execução e o uso de espaço, mantendo uma cópia de segurança sempre atualizada.

Numa primeira fase, foram desenvolvidos os scripts `backup_files.sh` e `backup.sh`. O primeiro *script* é responsável pela realização de *backups* de arquivos simples (sem subdiretorias) e atualiza os ficheiros com data de modificação posterior à do ficheiro correspondente no *backup*. Foi implementada a opção ***-c*** para um modo de verificação (*checking*), onde os comandos são exibidos sem realizar cópias. O segundo é uma extensão do anterior, suportando diretorias (que por sua vez podem ter subdiretorias) e permite a utilização de parâmetros como ***-b*** para ignorar certos ficheiros ou diretorias e ***-r*** para selecionar arquivos que correspondem a uma expressão regular.

A segunda fase englobou o desenvolvimento dos `scripts backup_summary.sh` e `backup_check.sh`, que complementam os primeiros com a geração de sumários para cada diretoria processada, detalhando o número de ficheiros copiados, atualizados, apagados e também quaisquer erros encontrados durante a execução. Por fim, o `backup_check.sh` verifica a integridade dos ficheiros do *backup* comparando o conteúdo com a diretoria de trabalho através do cálculo de *hash (md5sum)*, assegurando que o *backup* é uma réplica fiel do conteúdo da origem.

Capítulo 2

Metodologia

2.1 *backup_files.sh*

Este primeiro *script* realiza *backups* de ficheiros de uma diretoria fonte para uma diretoria de *backup*. Possui uma opção de *checking* de modo a verificar o que seria feito, sem realmente executar as ações de cópia ou exclusão.

```
if [[ $# -lt 2 ]]; then
    echo "Usage: $0 [-c] source_dir backup_dir"
    exit 1
fi
```

Figura 2.1: Verificação de argumentos na linha de comandos

Inicialmente é verificado se o número de argumentos passados para o *script* é menor do que 2. Se for, é exibida uma mensagem de uso e termina. A mensagem diz que o *script* espera dois argumentos, a diretoria fonte e a de *backup*, e opcionalmente pode ter um argumento ***-c***.

```
check_flag=false
if [[ "$1" == "-c" ]]; then
    check_flag=true
    shift
fi
```

Figura 2.2: Tratamento da opção **-c**

Se o primeiro argumento for ***-c***, o script define a variável `check_flag` e move o ponteiro de argumento (*shift*), ou seja, remove o ***-c*** da lista de argumentos.

```
source_dir="$1"
backup_dir="$2"

if [[ ! -d "$source_dir" ]]; then
    echo "Error: Source directory '$source_dir' does not exist."
    exit 1
fi

if [[ ! -d "$backup_dir" ]]; then
    if $check_flag; then
        echo "mkdir -p $backup_dir"
    else
        mkdir -p "$backup_dir"
        echo "mkdir -p $backup_dir"
    fi
fi
```

Figura 2.3: Validação das diretorias

Após o *shift*, caso o *script* seja passado com a opção de *checking*, o primeiro argumento é agora a diretoria de origem (`source_dir`) e o segundo argumento é a diretoria de *backup* (`backup_dir`). De seguida, é verificado se a diretoria de origem existe. Caso não exista, é exibida uma mensagem de erro e o *script* é encerrado. Por último, é verificado se a diretoria de *backup* existe. Se não existir e:

- Se o `check_flag` estiver ativo, o *script* apenas imprime o comando que criaria esta diretoria, mas não o executa.
- Se o `check_flag` não estiver ativo, o *script* cria a diretoria de *backup* e imprime o comando no terminal.

```
shopt -s dotglob

for file in "$source_dir"/*; do
    if [[ -f "$file" ]]; then
    filename=$(basename "$file")
    backup_file="$backup_dir/$filename"

        if [[ ! -e "$backup_file" || "$file" -nt "$backup_file" ]]; then
            echo "cp -a $file $backup_file"
            if ! $check_flag; then
                cp -a "$file" "$backup_file"
            fi
        fi
    fi
done
```

Figura 2.4: Configuração de opções globais para tratar arquivos ocultos/cópia de arquivos da diretoria de origem para a de *backup*

A opção `dotglob` faz com que os arquivos iniciados por pontos (arquivos ocultos) sejam incluídos nas expansões globais. Ou seja, ao listar os arquivos na diretoria de origem, arquivos ocultos também serão considerados.

De seguida, para cada arquivo na diretoria de origem (`source_dir`):

- É verificado se é um ficheiro regular (não uma diretoria);
- É extraído o nome base do arquivo com `basename` e é definido o caminho de destino da diretoria de *backup*;

- Se o arquivo de *backup* não existir ou o arquivo de origem for mais novo que o o de *backup*, o *script*:
  - Imprime o comando `cp -a`, que copiaria o arquivo;
  - Se o `check_flag` não estiver ativo, executa a cópia através de `cp -a`, preservando atributos como permissões e *timestamps*.

```
for file in "$backup_dir"/*; do
    filename=$(basename "$file")
    source_file="$source_dir/$filename"

    if [[ ! -e "$source_file" ]]; then
        echo "rm $file"
        if ! $check_flag; then
            rm "$file"
        fi
    fi
done

shopt -u dotglob
```

Figura 2.5: Remoção de arquivos na diretoria de backup que não existem mais na diretoria de origem/-Desativação da opção de arquivos ocultos

Por fim, para cada ficheiro na diretoria de *backup*, é verificado se o ficheiro correspondente não existe na diretoria de origem. Se assim for, o *script* imprime o comando `rm`, que removeria o ficheiro de *backup*. Caso a `check_flag` não esteja ativa, é executada a remoção do ficheiro. O último comando desativa a opção `dotglob`, restaurando o comportamento padrão onde arquivos ocultos não são incluídos nas expansões globais.

Em suma, o *script* sincroniza ficheiros entre uma diretoria origem e uma diretoria *backup*. Durante a execução, copia para o *backup* todos os ficheiros que estejam ausentes ou desatualizados em relação à origem e remove do *backup* quaisquer ficheiros que tenham sido eliminados na diretoria origem. Caso seja utilizada a opção de *checking*, o *script* opera em modo de simulação, apresentando as ações que seriam realizadas sem efetuar cópias ou remoções.

2.2 *backup.sh*

Como extensão do anterior, este *script* permite não só a realização *backups* de ficheiros, mas também de diretorias. Para além disso, adiciona novas opções e capacidades, como a exclusão de arquivos específicos e a filtragem por expressões regulares.

```
function usage() {
    echo "Usage: $0 [-c] [-b tfile] [-r regexpr] source_dir backup_dir"
    exit 1
}
```

Figura 2.6: Função *usage*

A função *usage* é utilizada para fornecer informações sobre como usar o *script* quando os argumentos passados são incorretos ou insuficientes. Exibe a sintaxe esperada do *script* e termina a execução com o comando `exit 1` se a utilização do *script* não seguir o formato adequado. O uso da função centraliza a lógica que exibe a mensagem de erro e termina a execução do *script*, facilitando o uso em diferentes pontos de validação e tornando o código mais limpo e reduzindo a duplicação.

```bash
check_flag=false
tfile=""
regex=""

while getopts 'cb:r:' flag; do
    case "${flag}" in
        c) check_flag=true ;;
        b) tfile="${OPTARG}" ;;
        r) regex="${OPTARG}" ;;
        *) usage ;;
    esac
done
shift $((OPTIND - 1))
```

Figura 2.7: Função *getopts* para processar *flags* e argumentos

Dentro do loop `while`, a função `getopts` processa cada uma das opções passadas na linha de comandos. Quando uma opção é encontrada, é atribuída à variável *flag*, que é então comparada com os casos definidos na estrutura `case`. Dependendo do valor de *flag*, o `case` executa o comando correspondente:

- Opção ***-c***: Ativa o modo de verificação (semelhante ao *script* anterior);
- Opção ***-b tfile***: Permite que o usuário forneça um arquivo contendo uma lista de ficheiros e/ou diretorias a serem excluídos do processo de *backup*;
- Opção ***-r regexpr***: Permite especificar uma expressão regular para filtrar os ficheiros a serem copiados;
- Quaisquer outras opções que não sejam válidas, a função *usage* é chamada.

No final, a linha `shift $((OPTIND - 1))` é usada para mover o ponteiro de parâmetros para que o *script* possa processar os dois últimos parâmetros restantes (`source_dir` e `backup_dir`).

```bash
if [[ $# -lt 2 ]]; then
    usage
fi

source_dir="$1"
backup_dir="$2"


if [[ ! -d "$source_dir" ]]; then
    echo "Error: Source directory '$source_dir' does not exist."
    exit 1
fi

shopt -s dotglob
```

Figura 2.8: Configuração de opções globais/Validação das diretorias

O processamento das diretorias assim como a opção `dotglob` é semelhante ao do *script* anterior, fazendo uso da função `usage` para manter o código compacto.

```
declare -A exclude_list
if [[ -n "$tfile" && -f "$tfile" ]]; then
    while IFS= read -r line; do
        exclude_list["$line"]=1S
    done < "$tfile"
fi
```

Figura 2.9: Configuração da lista de exclusão

Esta parte do *script* lida com a lista de arquivos que devem ser **excluídos** do *backup*. Se um arquivo for encontrado na diretoria de origem e o seu nome coincidir com uma entrada dessa lista, será ignorado durante o processo de *backup*:

- A linha `declare -A exclude_list` declara um ***array* associativo** chamado `exclude_list`. As chaves deste *array* serão os nomes dos arquivos a serem excluídos;
- A condição `if [[ -n "$tfile"&& -f "&tfile"]]; then` verifica duas coisas: se a variável ***tfile*** não está vazia (ou seja, um ficheiro foi fornecido), e verifica se ***tfile*** é um **ficheiro regular** (ou seja, o ficheiro existe e não é uma diretoria ou um link). Se ambas as condições forem verdadeiras, o *script* irá continuar para ler o conteúdo do arquivo de exclusão;
- A linha `while IFS= read -r line; do` inicia um ***loop while*** que lê o arquivo ***tfile*** linha por linha. `IFS=` garante que nenhum espaço em branco à esquerda ou à direita seja removido de cada linha, enquanto `-r` impede que as barras invertidas nas linhas sejam interpretadas como caracteres de escape;
- Dentro do ***loop***, para cada linha (que representa o nome de um ficheiro/diretoria a ser excluído), o *script* adiciona uma entrada no *array* associativo `exclude_list`. A **chave** do *array* é o nome do ficheiro/diretoria `$line`, e o valor é configurado como `1S`, arbitrário. Quando o *loop* chega ao final do arquivo, é interrompido.

```
function recursive_backup() {
    local current_source_dir="$1"
    local current_backup_dir="$2"

    if [[ ! -d "$current_backup_dir" ]]; then
        if $check_flag; then
            echo "mkdir -p $current_backup_dir"
        else
            mkdir -p "$current_backup_dir"
            echo "mkdir -p $current_backup_dir"
        fi
    fi
```

Figura 2.10: Função principal de *backup* recursivo

Neste contexto, a recursividade é usada através da função `recursive_backup()` para realizar o ***backup* de diretorias e subdiretorias** de forma hierárquica. Quando a função encontra uma subdiretoria, ela chama-se a si mesma para processar o seu conteúdo, até que não haja mais subdiretorias a serem processadas. Desta forma, é importante definir as variáveis localmente, para garantir que cada instância

da função tenha o seu próprio conjunto de váriáveis, neste caso, as diferentes subdiretorias. A validação das diretorias de *backup* surge de forma idêntica ao primeiro *script*, mas desta vez incluída na função recursiva, para garantir que são processadas hierarquicamente.

```
    for file in "$current_source_dir"/*; do
        local filename=$(basename "$file")

        if [[ -n "${exclude_list[$filename]}" ]]; then
            continue
        fi
```

Figura 2.11: Percorrer os arquivos na diretoria origem

Neste ciclo começa a parte principal do *script*. A função começa por percorrer todos os ficheiros e subdiretorias dentro de `current_source_dir` (que pode ser a diretoria de origem ou uma subdiretoria durante a recursão). A variável `filename` armazena apenas o nome do ficheiro ou diretoria, usando o comando `basename`. Se o nome do arquivo estiver na lista de exclusões (`exclude_list`), é ignorado.

```
    local backup_file="$current_backup_dir/$filename"

    if [[ -f "$file" && ( -z "$regex" || "$filename" =~ $regex ) ]]; then
        if [[ ! -e "$backup_file" ]]; then
            echo "cp -a $file $backup_file"
            if ! $check_flag; then
                cp -a "$file" "$backup_file"
            fi

        elif [[ "$file" -nt "$backup_file" || $(cmp -s "$file" "$backup_file" || echo "different") == "different" ]]; then
            if ! check_flag; then
                echo "cp -a $file $backup_file"
                cp -a $file $backup_file
            fi
        fi
```

Figura 2.12: *Backup* de arquivos

De seguida, a condição verifica se o item é um ficheiro regular. Se `regex` não for fornecido, o *script* copia todos os ficheiros. Caso contrário, ele copia apenas os ficheiros cujos nomes correspondem ao *regex* fornecido. Se o ficheiro ainda não existir no *backup*, é copiado. Caso este seja mais recente na origem que no *backup* ou o seu conteúdo seja diferente, também será feita a cópia de segurança, isto sempre de acordo com a *flag* `check_flag`.

```
        elif [[ -d "$file" ]]; then
            local has_matching_files=false
            if [[ -n "$regex" ]]; then
                for subfile in "$file"/*; do
                    local subfilename=$(basename "$subfile")
                    if [[ -f "$subfile" && "$subfilename" =~ $regex ]]; then
                        has_matching_files=true
                        break
                    fi
                done
            else
                has_matching_files=true
            fi

            if $has_matching_files; then
                recursive_backup "$file" "$backup_file"
            fi
        fi
    done
```

Figura 2.13: *Backup* de subdiretorias

No caso do item ser uma diretoria, a variável `has_matching_files`, que será usada para verificar se algum arquivo na subdiretoria corresponde ao `regex` (se fornecido), é inicialmente definida como *false*.

- Se `regex` for fornecido, o *script* percorre todos os ficheiros dentro da subdiretoria e verifica se algum arquivo corresponde ao `regex`. Caso encontre algum ficheiro que combine, `has_matching_files` é definido como *true* e a busca é interrompida com *break*;

- Se `regex` não for fornecido, qualquer diretoria é considerada para *backup*, independentemente dos ficheiros que contenha.

Se `has_matching_files` for *true*, o *script* chama a função de forma recursiva para a subdiretoria, o que significa que irá fazer o *backup* do conteúdo no seu interior.

```
    for backup_file in "$current_backup_dir"/*; do
        local filename=$(basename "$backup_file")
        local source_file="$current_source_dir/$filename"

        if [[ ! -e "$source_file" && -e "$backup_file" ]]; then
            echo "rm -r $backup_file"
            if ! $check_flag; then
                rm -r "$backup_file"
            fi
        fi
    done
}
```

Figura 2.14: Remoção de arquivos de *backup* obsoletos

Neste ciclo, o *script* verifica se há arquivos na diretoria de *backup* que não existem mais na diretoria de origem. Se não existir correspondência, o arquivo obsoleto será removido, de acordo com as condições de *checking* já explicadas.

2.3 *backup_summary.sh*

De forma a complementar o processo de *backup* já densenvolvido, este *script* é uma aprimoração do anterior, que agora inclui a contabilização de **erros, avisos, arquivos copiados, atualizados e excluídos** durante o processo. Além disso, mantém o controlo sobre o **tamanho dos arquivos copiados e eliminados**.

```
warnings=0
errors=0
updated=0
copied=0
deleted=0
total_size_copied=0
total_size_deleted=0
```

Figura 2.15: Declaração das variáveis globais

Para contabilizar todos estes parâmetros, no início do *script* são declaradas as variáveis globais.

```
    local local_warnings=0
    local local_errors=0
    local local_updated=0
    local local_copied=0
    local local_deleted=0
    local local_size_copied=0
    local local_size_deleted=0
```

Figura 2.16: Declaração das variáveis locais

Dentro da função recursiva, também são declaradas variáveis locais equivalentes para que possamos acompanhar as operações por cada chamada recursiva e somar ao total no final de cada execução, o que permite uma visão específica de cada diretoria ao ser processada.

```
if [[ ! -d "$current_backup_dir" ]]; then
    if $check_flag; then
        echo "mkdir -p $current_backup_dir"
    else
        mkdir -p "$current_backup_dir" || { echo "Error creating directory: $current_backup_dir"; local_errors=$((local_errors + 1)); }
        echo "mkdir -p $current_backup_dir"
    fi
fi
```

Figura 2.17: Tratamento de erros na criação de diretorias

```
    if [[ -f "$file" && ( -z "$regex" || "$filename" =~ $regex ) ]]; then
        if [[ ! -e "$backup_file" ]]; then
            echo "cp -a $file $backup_file"
            if ! $check_flag; then
                cp -a "$file" "$backup_file" || { echo "Error copying file: $file"; local_errors=$((local_errors + 1)); continue; }
                local_copied=$((local_copied + 1))
                local_size_copied=$((local_size_copied + $(stat -c%s "$file")))
            fi
        elif [[ "$file" -nt "$backup_file" || $(cmp -s "$file" "$backup_file" || echo "different") == "different" ]]; then
            if ! $check_flag; then
                echo "cp -a $file $backup_file"
                cp -a "$file" "$backup_file" || { echo "Error copying file: $file"; local_errors=$((local_errors + 1)); continue; }
                local_updated=$((local_updated + 1))
            fi
        fi

        if [[ "$backup_file" -nt "$file" ]]; then
            echo "WARNING: backup entry $backup_file is newer than $file; Should not happen"
            local_warnings=$((local_warnings + 1))
        fi
```

Figura 2.18: Tratamento de erros na cópia de arquivos/*backups* mais novos

```
for backup_file in "$current_backup_dir"/*; do
    local filename=$(basename "$backup_file")
    local source_file="$current_source_dir/$filename"

    if [[ ! -e "$source_file" && -e "$backup_file" ]]; then
        echo "rm -r $backup_file"
        if ! $check_flag; then
            rm -r "$backup_file" || { echo "Error removing $backup_file"; local_errors=$((local_errors + 1)); continue; }
            local_deleted=$((local_deleted + 1))
            local_size_deleted=$((local_size_deleted + $(stat -c%s "$backup_file" 2>/dev/null || echo 0)))
        fi
    fi
done
```

Figura 2.19: Tratamento de erros na exclusão de arquivos

Foram incluídas verificações de erro para as operações de criação de diretorias, cópia e exclusão de arquivos. Caso ocorra um erro, ele é registado e o processo continua sem tentar novamente para aquele ficheiro/diretoria.

```
echo "While backing up $current_source_dir: $local_errors Errors; $local_warnings Warnings;
$local_updated Updated; $local_copied Copied ($local_size_copied B); $local_deleted Deleted ($local_size_deleted B)"
```

Figura 2.20: Resumo das operações em cada diretoria

Para cada diretoria processada, o *script* imprime o resumo das operações (erros, avisos, arquivos copiados, eliminados, atualizados e o tamanho total copiado/eliminado)

```
    warnings=$((warnings + local_warnings))
    errors=$((errors + local_errors))
    updated=$((updated + local_updated))
    copied=$((copied + local_copied))
    deleted=$((deleted + local_deleted))
    total_size_copied=$((total_size_copied + local_size_copied))
    total_size_deleted=$((total_size_deleted + local_size_deleted))
}
```

Figura 2.21: Acúmulo total

No final da execução da função, as variáveis locais são somadas às globais, acumulando os totais gerais. Estas modificações permitem que o *script* forneça uma visão detalhada de cada operação realizada durante o *backup*.

2.4 *backup_check.sh*

Por fim, este último *script* verifica se os ficheiros numa diretoria de *backup* correspondem aos ficheiros numa diretoria de origem, fazendo uma comparação de integridade usando *hashes MD5* e destacando a difrença entre os dois. Esta função de resumo criptográfico transforma qualquer entrada de dados numa sequência de caracteres de comprimento fixo. Como o *hash* é único para cada conteúdo, pode ser usado como uma "assinatura digital"para identificar ficheiros. A implementação desta função utiliza funcionalidades já descritas.

```
        if [[ -f "$file" && -f "$backup_file" ]]; then
            local source_hash=$(md5sum "$file" | awk '{print $1}')
            local backup_hash=$(md5sum "$backup_file" | awk '{print $1}')
            if [[ "$source_hash" != "$backup_hash" ]]; then
                echo "$file and $backup_file differ."
            fi
```

Figura 2.22: Comparação de arquivos usando MD5

Se um item é um ficheiro tanto na diretoria de origem quanto na de *backup*, o *script* calcula o *hash* MD5 de ambos e compara. Se os *hashes* diferem, é imprimida uma mensagem informando a diferença.

```
        elif [[ -d "$file" && -d "$backup_file" ]]; then
            recursive_check "$file" "$backup_file"
```

Figura 2.23: Verificação de subdiretorias

Se o item é uma diretoria tanto na diretoria de origem quanto na de *backup*, a função `recursive_check` é chamada novamente, passando essas diretorias como parâmetros para verificar os seus conteúdos. Essa chamada recursiva permite que o script analise a estrutura de subdiretorias de forma completa.

```
        elif [[ -e "$file" && ! -e "$backup_file" ]]; then
            echo "File or directory $file is missing in backup."

        elif [[ ! -e "$file" && -e "$backup_file" ]]; then
            echo "File or directory $backup_file is missing in source."
        fi
    done
}
```

Figura 2.24: Verificação de itens ausentes

Caso o item exista em `source_dir` mas não em `backup_dir`, o *script* imprime uma mensagem indicando que o item está ausente no *backup*, e vice-versa.

Capítulo 3

Testes

Este capítulo descreve o processo de testes realizado para validar o comportamento e a integridade dos *scripts* desenvolvidos. O seu principal objetivo é garantir que as funcionalidades principais do sistema, como a cópia, exclusão e verificação de integridade de arquivos e diretorias funcionam de forma consistente.

3.1 Testes de Configuração e Pré-Requesitos

```
zval@zval:~/Desktop/Trabalho 1$ ./backup.sh .
Usage: ./backup.sh [-c] [-b tfile] [-r regexpr] source_dir backup_dir
zval@zval:~/Desktop/Trabalho 1$ ./backup.sh /home/source /home/BACKUP
Error: Source directory '/home/source' does not exist.
```

Figura 3.1: Verificação de argumentos insuficientes e Diretoria de origem inexistente

3.2 Testes de *Backup* e sincronização de Arquivos

```
zval@zval:~/Desktop/Trabalho 1$ ./backup.sh . /home/zval/Desktop/BACKUP
mkdir -p /home/zval/Desktop/BACKUP
cp -a ./backup_check.sh /home/zval/Desktop/BACKUP/backup_check.sh
cp -a ./backup_files.sh /home/zval/Desktop/BACKUP/backup_files.sh
cp -a ./backup.sh /home/zval/Desktop/BACKUP/backup.sh
cp -a ./backup_summary.sh /home/zval/Desktop/BACKUP/backup_summary.sh
cp -a ./teste.txt /home/zval/Desktop/BACKUP/teste.txt
```

Figura 3.2: *Backup* de ficheiros novos

```
zval@zval:~/Desktop/Trabalho 1$ ./backup.sh . /home/zval/Desktop/BACKUP
cp -a ./teste.txt /home/zval/Desktop/BACKUP/teste.txt
```

Figura 3.3: *Backup* de ficheiros modificados

```
zval@zval:~/Desktop/Trabalho 1$ ./backup.sh . /home/zval/Desktop/BACKUP
rm -r /home/zval/Desktop/BACKUP/teste.txt
```

Figura 3.4: Exclusão de ficheiros obsoletos

```
zval@zval:~/Desktop/Trabalho 1$ touch .hidden.txt
zval@zval:~/Desktop/Trabalho 1$ mkdir .Diretoria
zval@zval:~/Desktop/Trabalho 1$ ./backup.sh . /home/zval/Desktop/BACKUP
mkdir -p /home/zval/Desktop/BACKUP/.Diretoria
cp -a ./.hidden.txt /home/zval/Desktop/BACKUP/.hidden.txt
```

Figura 3.5: Preservação de Diretorias e Ficheiros escondidos

```
zval@zval:~/Desktop/Trabalho 1$ ./backup.sh -c . /home/zval/Desktop/BACKUP
mkdir -p /home/zval/Desktop/BACKUP
cp -a ./backup_check.sh /home/zval/Desktop/BACKUP/backup_check.sh
cp -a ./backup_files.sh /home/zval/Desktop/BACKUP/backup_files.sh
cp -a ./backup.sh /home/zval/Desktop/BACKUP/backup.sh
cp -a ./backup_summary.sh /home/zval/Desktop/BACKUP/backup_summary.sh
mkdir -p /home/zval/Desktop/BACKUP/.Diretoria
cp -a ./.hidden.txt /home/zval/Desktop/BACKUP/.hidden.txt
cp -a ./teste.txt /home/zval/Desktop/BACKUP/teste.txt
```

Figura 3.6: *Flag* de verificação `-c`

```
zval@zval:~/Desktop/Trabalho 1$ ./backup.sh -r '.sh' . /home/zval/Desktop/BACKUP
mkdir -p /home/zval/Desktop/BACKUP
cp -a ./backup_check.sh /home/zval/Desktop/BACKUP/backup_check.sh
cp -a ./backup_files.sh /home/zval/Desktop/BACKUP/backup_files.sh
cp -a ./backup.sh /home/zval/Desktop/BACKUP/backup.sh
cp -a ./backup_summary.sh /home/zval/Desktop/BACKUP/backup_summary.sh
```

Figura 3.7: Filtro de expressão regular `-r`

```
zval@zval:~/Desktop/Trabalho 1$ echo -e "teste.txt\n.hidden.txt" > tfile.txt
zval@zval:~/Desktop/Trabalho 1$ ./backup.sh -b 'tfile.txt' . /home/zval/Desktop/BACKUP
mkdir -p /home/zval/Desktop/BACKUP
cp -a ./backup_check.sh /home/zval/Desktop/BACKUP/backup_check.sh
cp -a ./backup_files.sh /home/zval/Desktop/BACKUP/backup_files.sh
cp -a ./backup.sh /home/zval/Desktop/BACKUP/backup.sh
cp -a ./backup_summary.sh /home/zval/Desktop/BACKUP/backup_summary.sh
mkdir -p /home/zval/Desktop/BACKUP/.Diretoria
cp -a ./tfile.txt /home/zval/Desktop/BACKUP/tfile.txt
```

Figura 3.8: Lista de exclusão `-b`

```
zval@zval:~/Desktop/Trabalho 1$ ./backup_summary.sh . /home/zval/Desktop/BACKUP
mkdir -p /home/zval/Desktop/BACKUP
cp -a ./backup_check.sh /home/zval/Desktop/BACKUP/backup_check.sh
cp -a ./backup_files.sh /home/zval/Desktop/BACKUP/backup_files.sh
cp -a ./backup.sh /home/zval/Desktop/BACKUP/backup.sh
cp -a ./backup_summary.sh /home/zval/Desktop/BACKUP/backup_summary.sh
mkdir -p /home/zval/Desktop/BACKUP/.Diretoria
While backuping ./.Diretoria: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
cp -a ./.hidden.txt /home/zval/Desktop/BACKUP/.hidden.txt
cp -a ./teste.txt /home/zval/Desktop/BACKUP/teste.txt
cp -a ./tfile.txt /home/zval/Desktop/BACKUP/tfile.txt
While backuping .: 0 Errors; 0 Warnings; 0 Updated; 7 Copied (10245B); 0 Deleted (0B)
```

Figura 3.9: *Backup* com sumário

3.3 Testes de Validação de Integridade

```
zval@zval:~/Desktop/Trabalho 1$ ./backup_check.sh . /home/zval/Desktop/BACKUP/
zval@zval:~/Desktop/Trabalho 1$
```

Figura 3.10: Verificação de Diretorias de origem e *backup* idênticas

```
zval@zval:~/Desktop/Trabalho 1$ echo -e "teste alterado" > teste.txt
zval@zval:~/Desktop/Trabalho 1$ touch novoteste.txt
zval@zval:~/Desktop/Trabalho 1$ ./backup_check.sh . /home/zval/Desktop/BACKUP/
File or directory ./novoteste.txt is missing in backup.
./teste.txt and /home/zval/Desktop/BACKUP//teste.txt differ.
```

Figura 3.11: Verificação com ficheiros diferentes/ausentes

3.4 Testes de Função Recursiva com Estruturas Complexas de Diretorias

```
~/test_backup
├── .hidden_dir
│   ├── .config_hidden
│   └── .data_hidden
├── configs
│   ├── defaults
│   │   └── .default_settings
│   └── settings
│       ├── config.yaml
│       └── .hidden_settings
│           └── .secret_config
├── docs
│   ├── .hidden_notes.txt
│   ├── letters
│   │   ├── letter1.txt
│   │   └── .letter_hidden.txt
│   ├── readme.txt
│   └── reports
│       ├── report1.txt
│       └── report2.txt
```

Figura 3.12: Estrutra hierárquica complexa de Diretorias e Ficheiros

```
zval@zval:~/Desktop/Trabalho 1$ ./backup.sh ~/test_backup /home/zval/Desktop/BACKUP
mkdir -p /home/zval/Desktop/BACKUP
mkdir -p /home/zval/Desktop/BACKUP/configs
mkdir -p /home/zval/Desktop/BACKUP/configs/defaults
cp -a /home/zval/test_backup/configs/defaults/.default_settings /home/zval/Desktop/BACKUP/configs/defaults/.default_settings
mkdir -p /home/zval/Desktop/BACKUP/configs/settings
cp -a /home/zval/test_backup/configs/settings/config.yaml /home/zval/Desktop/BACKUP/configs/settings/config.yaml
mkdir -p /home/zval/Desktop/BACKUP/configs/settings/.hidden_settings
cp -a /home/zval/test_backup/configs/settings/.hidden_settings/{.secret_config} /home/zval/Desktop/BACKUP/configs/settings/.hidden_settings/{.secret_config}
mkdir -p /home/zval/Desktop/BACKUP/docs
cp -a /home/zval/test_backup/docs/.hidden_notes.txt /home/zval/Desktop/BACKUP/docs/.hidden_notes.txt
mkdir -p /home/zval/Desktop/BACKUP/docs/letters
cp -a /home/zval/test_backup/docs/letters/letter1.txt /home/zval/Desktop/BACKUP/docs/letters/letter1.txt
cp -a /home/zval/test_backup/docs/letters/.letter_hidden.txt /home/zval/Desktop/BACKUP/docs/letters/.letter_hidden.txt
cp -a /home/zval/test_backup/docs/readme.txt /home/zval/Desktop/BACKUP/docs/readme.txt
mkdir -p /home/zval/Desktop/BACKUP/docs/reports
cp -a /home/zval/test_backup/docs/reports/report1.txt /home/zval/Desktop/BACKUP/docs/reports/report1.txt
cp -a /home/zval/test_backup/docs/reports/report2.txt /home/zval/Desktop/BACKUP/docs/reports/report2.txt
mkdir -p /home/zval/Desktop/BACKUP/.hidden_dir
cp -a /home/zval/test_backup/.hidden_dir/.config_hidden /home/zval/Desktop/BACKUP/.hidden_dir/.config_hidden
cp -a /home/zval/test_backup/.hidden_dir/.data_hidden /home/zval/Desktop/BACKUP/.hidden_dir/.data_hidden
```

Figura 3.13: *Backup* de todos os ficheiros e diretorias da estrutura

3.5 Testes Fornecidos

Teste fornecido no *elearning* pelo docente:

```
zval@zval:~/Desktop/SO/Trabalho 1/TESTE$ ./test_a1.sh
Score: 100
```

Figura 3.14: Teste fornecido com uma pontuação de 100

Capítulo 4

Conclusão

Este projeto tinha como objetivo o desenvolvimento de *scripts* para automatizar a criação e atualização de cópias de segurança de ficheiros e diretorias de forma eficiente e prática. A solução abrange diferentes aspetos de um processo de *backup*, desde a cópia de ficheiros e tratamento recursivo de diretorias e subdiretorias até a funcionalidades adicionais como filtragem por expressão regular e exclusão de arquivos específicos. Foi implementada uma verificação de integridade, com a comparação de *hashes*, assegurando que as cópias sejam fiés ao conteúdo original. Foi também implementado um sistema de relatório detalhado, que contabiliza erros, arquivos copiados, atualizados e excluídos que oferece uma visão completa das operações realizadas. Além disso, a opção de execução em modo de *checking* permite que o utilizador reveja as ações a serem executadas antes da execução real, proporcionando uma maior segurança ao processo. Todos os objetivos propostos foram desenvolvidos com sucesso, validados por testes que foram elaborados de modo a abranger todoas as funcionalidades possíveis que o processo de *backup* oferece.

Capítulo 5

Bibliografia

Bibliografia utilizada:

- https://stackoverflow.com/
- https://askubuntu.com/
- https://chatgpt.com/