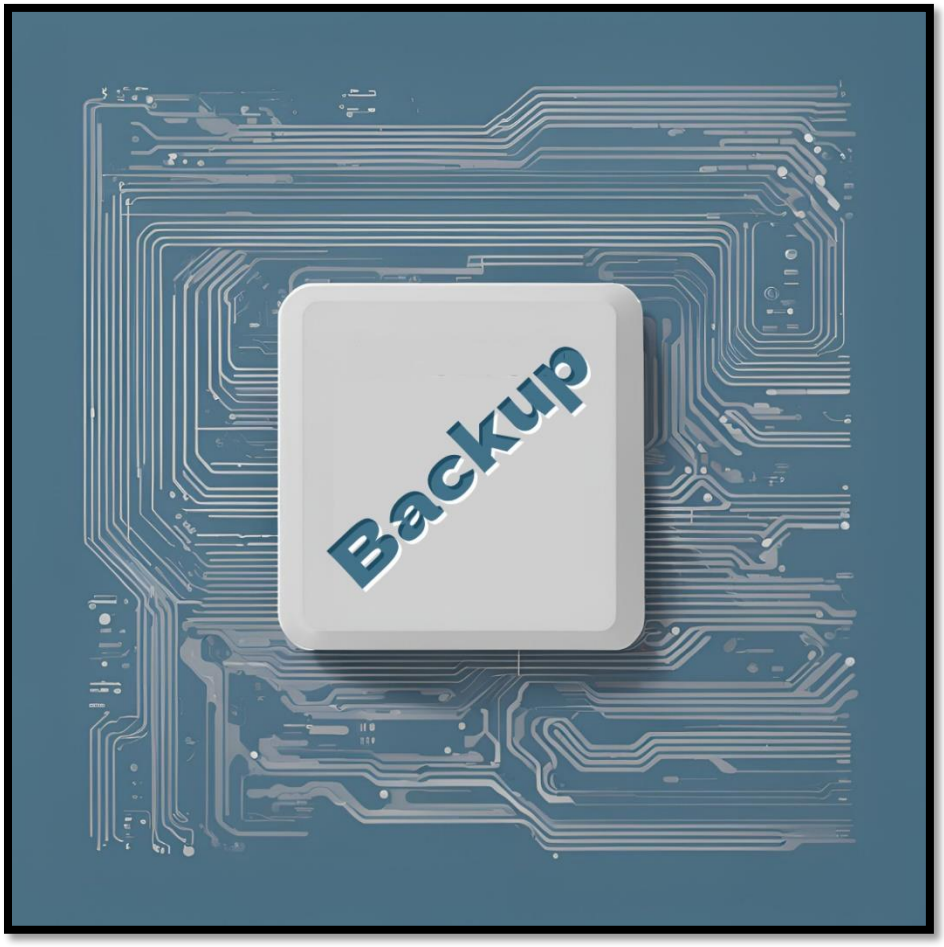
Trabalho 1 [Sistemas Operativos]

Ferramenta de criação/atualização de cópias de segurança em bash



Curso: Engenharia Informática

Realizado por: Paulo Cunha [NMec: 118741] e Rafael Ferreira [NMec: 118803]

Índice

Introdução

No âmbito da disciplina de Sistemas Operativos, foi-nos proposto como primeiro trabalho prático, o desenvolvimento de uma ferramenta que possibilitasse a criação/atualização de cópias de segurança (backup) em bash.

Utilizando comandos simples de bash e alguns métodos como iterações, recursividade e mais, conseguimos construir scripts funcionais e minimamente organizados.

O funcionamento desta ferramenta de backup está dependente do bom funcionamento e implementação de funções criadas por nós através de pesquisa e de conhecimentos apreendidos nas aulas práticas e teóricas de Sistemas Operativos.

Este relatório tem como objetivo explicar a forma com que organizamos/estruturamos e pensamos no código implementado para o bom funcionamento da ferramenta em questão.

O projeto foi desenvolvido da seguinte forma: os exercícios 1 desenvolvido por ambos, 2 e 3 feito por Paulo Cunha [118741] e o 4 desenvolvido por Rafael Ferreira [118803]. Decidimos desta forma, pois achamos que seria muito confuso trabalhar em simultâneo no 2 já que era o mais complexo e como do exercício 2 para o 3 podíamos reaproveitar bastante código não seria demasiado cansativo para apenas uma pessoa resolver os dois.

Objetivos

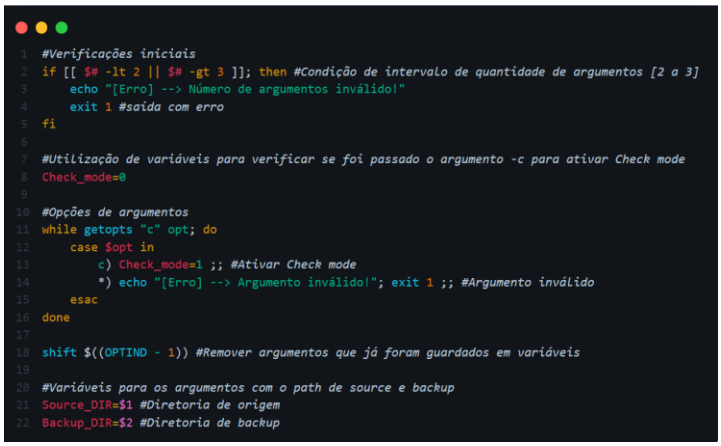
1. Desenvolvimento de scripts seguindo parâmetros e estruturas obrigatórias;
2. Desenvolver 4 scripts com diferentes abordagens ao problema;
    2.1. Backup_files.sh: Script que considera a apenas a diretoria fonte e que vai fazer backup apenas de ficheiros nela incluídos ignorando subdiretorias presentes. Pode ter "-c" como parâmetro extra (onde, o mesmo, ativa o modo checking que vai apenas apresentar os comandos que seriam executados na consola);
    2.2. Backup.sh: Script semelhante ao anterior, mas agora considerando também a possibilidade de haver subdiretorias dentro da diretoria fonte. Neste caso consideram-se, para além do parâmetro "-c", também o "-b tfile" (permite a indicação de um ficheiro de texto que contém uma lista de ficheiros/diretorias que não devem ser copiados) e o "-r regexpr" (indica que apenas devem ser copiados os ficheiros que verificam a expressão regular passada);
    2.3. Backup_summary.sh: Script que tem funcionamento basicamente idêntico a backup.sh, mas com a particularidade de ser escrito na consola um sumário do backup feito, onde há a indicação do número de possíveis erros que possam acontecer, a contagem de quantos ficheiros forma copiados e removidos e o tamanho total em bytes desses ficheiros;
    2.4. Backup_check.sh: Script que verifica se o conteúdo dos ficheiros na diretoria de backup é igual ao conteúdo dos ficheiros correspondentes na diretoria de trabalho. Deve implementar o comando md5sum, escrevendo uma mensagem se forem iguais ou não e se houver erros durante a comparação.
3. Em todos os scripts a estrutura da linha de comando tem de ser respeitada, os ficheiros devem ser copiados com o comando cp, ficheiros mais antigos presentes em backup (caso esta já exista) devem ser atualizados e se já não existirem na diretoria de trabalho devem ser removidos (subdiretorias incluídas).
4. Desenvolver capacidades de resolução de problemas através de código bem estruturado e funcional, respeitando normas e boas práticas. Aplicando conhecimentos de código bash apreendido em SO.

O Código

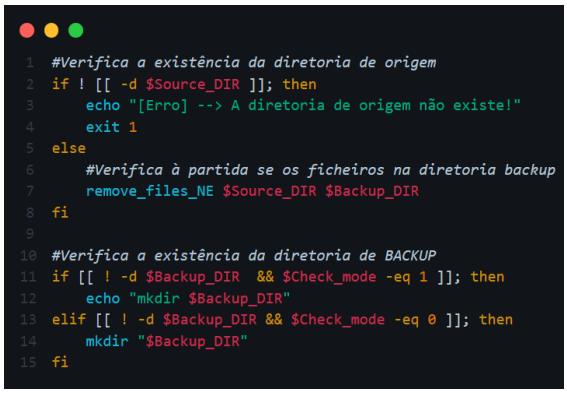
1-Backup_files.sh)

1. Definição de condições básicas para bom funcionamento do script:

1.1. Numa primeira fase comecei por fazer as verificações necessárias para a boa inicialização do script, definindo o número de argumentos que podem ser passados ao script através da estrutura condicional if e do contador de argumentos $#, neste caso são obrigatórios dois argumentos (o path para a diretoria de origem (fonte) e o path para a diretoria de destino (backup)) e opcionalmente pode ser passado o argumento "-c" totalizando três argumentos no máximo, o script termina imediatamente a sua execução com erro "exit 1" se este não tiver o número de argumentos adequado; Para lidar com o problema de o argumento opcional interferir com a ordem de armazenamento dos paths em $1 e $2, primeiro defini uma variável para o check e utilizei o comando getops para controlar os argumentos passados e se for passado "-c" a variável check fica ativa (1), controlando também a possibilidade de serem passados argumentos inválidos, depois disso utilizo o comando shift para remover os argumentos do getops e assim garantir que $1 recebe a diretoria fonte e $2 recebe a diretoria backup a parte (optind - 1) que calcula quantas vezes o Shift será feito, já que a variável interna optind guarda o índex do primeiro argumento não processado pelo getops;



```bash
#Verificações iniciais
if [[ $# -lt 2 || $# -gt 3 ]]; then #Condição de intervalo de quantidade de argumentos [2 a 3]
    echo "[Erro] --> Número de argumentos inválido!"
    exit 1 #saída com erro
fi

#Utilização de variáveis para verificar se foi passado o argumento -c para ativar Check mode
Check_mode=0

#Opções de argumentos
while getopts "c" opt; do
    case $opt in
        c) Check_mode=1 ;; #Ativar Check mode
        *) echo "[Erro] --> Argumento inválido!"; exit 1 ;; #Argumento inválido
    esac
done

shift $((OPTIND - 1)) #Remover argumentos que já foram guardados em variáveis

#Variáveis para os argumentos com o path de source e backup
Source_DIR=$1 #Diretoria de origem
Backup_DIR=$2 #Diretoria de backup
```



```bash
#Verifica a existência da diretoria de origem
if ! [[ -d $Source_DIR ]]; then
    echo "[Erro] --> A diretoria de origem não existe!"
    exit 1
else
    #Verifica à partida se os ficheiros na diretoria backup
    remove_files_NE $Source_DIR $Backup_DIR
fi

#Verifica a existência da diretoria de BACKUP
if [[ ! -d $Backup_DIR  && $Check_mode -eq 1 ]]; then
    echo "mkdir $Backup_DIR"
elif [[ ! -d $Backup_DIR && $Check_mode -eq 0 ]]; then
    mkdir "$Backup_DIR"
fi
```

1.2. Verificação da existência das diretorias fonte e backup através da estrutura condicional if. No caso da diretoria fonte se esta não existir a execução deve terminar com mensagem de erro, já se a diretoria backup não existir esta deve ser criada através do comando mkdir.

2. Parte principal do funcionamento do script:
   2.1. Dividi a execução em duas partes com um if, caso o check mode esteja ativo o código irá funcionar exatamente do mesmo modo, mas sem executar comandos;

```
#Iterar sobre os ficheiros para fazer o backup a partir do cp -a (comando copy)
for file in "$Source_DIR"/{*,.*}; do
    if [[ -d $file ]]; then #Ignorar diretórios
        continue
    fi
```

   2.2. Para realizar o backup utilizei a estrutura de repetição for, mas filtrei para iterar por todo o tipo de ficheiros utilizando a pattern "/{*,.*}", onde (/) significa raiz da diretoria, (*) significa todas as subdiretorias e ficheiros visíveis e (.*) todas as subdiretorias e ficheiros escondidos;

   2.3. Na iteração é feita outra filtragem onde serão verificados apenas os ficheiros, estes por sua vez, serão verificados quanto à sua existência na diretoria backup, onde se não existirem serão copiados utilizando o comando "cp -a" (-a preserva a data de modificação), caso contrário o ficheiro da fonte é comparado com o existente em backup para verificar qual possui a data de modificação mais recente através do operador -nt (newer than) e se o ficheiro origem for mais recente então remove o ficheiro que está em backup com o comando rm e copia o da fonte para backup com o comando cp -a, se não apenas irá avisar que não irá copiar devido ao ficheiro presente em backup ser mais recente. Todos os comandos de rm e cp possuem uma condição onde se não se conseguir executar o comando eles passam esse ficheiro e mostram uma mensagem de erro.

```
    if [[ $Check_mode -eq 1 ]]; then #Exucução do programa de acordo com o argumento -c (Apenas imprime comandos que se
        echo "WARNING: Versão do ficheiro encontrada em backup desatualizada [Subistituir]"

        echo "rm  $Backup_DIR/$filename"

if [[ -e "$Backup_DIR/$filename" ]]; then
        if [[ "$file" -nt "$Backup_DIR/$filename" ]]; then
        rm  "$Backup_DIR/$filename" || { echo "[ERRO] ao remover $Backup_DIR/$filename"; continue; } #Remover ficheiro
        log $log_file "rm "$Backup_DIR/$filename"" #Registo do log

        cp -a $file $Backup_DIR || { echo "[ERRO] ao copiar $file para $Backup_DIR"; continue; } #Cópia do ficheiro
        log $log_file "cp -a $file $Backup_DIR"

        echo "$filename substituído"
    fi
else
    echo "WARNING: Backup possui versão mais recente do ficheiro $file --> [Não copiado]" #Mensagem de aviso
    log $log_file "Warning não substituído"
fi
```

```
    else
        echo "WARNING: Backup possui versão mais recente do ficheiro $file --> [Não copiado]" #Mensagem de aviso
        log $log_file "Warning não substituído"
    fi
  else
    if [[ $Check_mode -eq 1 ]]; then
        echo "cp -a $file $Backup_DIR"
    else
        cp -a "$file" "$Backup_DIR" || { echo "[ERRO] ao copiar $file para $Backup_DIR"; continue; } #Cópia do ficheiro
        log "$log_file" "cp -a $file $Backup_DIR"

        echo $log_file "[Ficheiro $file copiado para backup]" #Mensagem de sucesso
    fi
  fi
one
```

```
if [[ ! -e "$source_file" ]]; then
    echo "A remover $backup_file [não existe em $source_dir]"
    rm "$backup_file" || { echo "[ERRO] ao remover $backup_file"; } #Remover ficheiro
    log $log_file "rm "$backup_file""
fi
```

3. Uma verificação implementada mais tardiamente, mas que deve ser feita, que consiste na verificação da existência de ficheiros na diretoria backup que não estão presentes na diretoria fonte, estes devem ser apagados da diretoria backup com o comando rm, isto implicou o desenvolvimento de uma função de nome **remove_files_NE** explicada mais em detalhe em Funções/scripts auxiliares [17];

4. Implementações extra:
   4.1. Também o funcionamento do script **function_log.sh** está presente em Funções/scripts auxiliares [17]. Tem por objetivo guardar um ficheiro .log que possui informações sobre o backup.

2-Backup.sh)

1. Definição de condições básicas para bom funcionamento do script: Esta parte inicial é exatamente igual ao realizado no script backup_files.sh, mas desta vez o intervalo de argumentos é maior já que podemos passar sem contar com os obrigatórios mais três argumentos ("-c", "-b" e "-r", ambos com optarg).

   1.1. Modificações aplicadas na definição da utilização do comando getops. Adicionei os dois argumentos novos: o "-b" que recebe como optarg o nome do ficheiro de texto onde estão apresentados os nomes dos ficheiros/diretorias que devem ser ignorados durante o backup e o "-r" que recebe como optarg uma expressão regex que será usada para apenas fazer copia de ficheiros que respeitam a expressão regex. Para contornar o mesmo problema de argumentos apenas defini mais duas variáveis para guardar os valores de cada optarg e na definição dos argumentos no getops para indicar que possuem optarg a ser passado usei ":". Depois de já não serem precisos os argumentos estes são removidos com o comando shift.

```
#Utilização de variáveis para argumentos
Check_mode=0
file_title=""
regexpr=""

#Opções de argumentos
while getopts "cb:r:" opt; do
    case $opt in
        c) Check_mode=1 ;;
        b) file_title="$OPTARG" ;;
        r) regexpr="$OPTARG" ;;
        \?) echo "[Erro] --> Opção inválida: -$OPTARG"; exit 1 ;;
        :) echo "[Erro] --> A opção -$OPTARG requer um argumento."; exi
    esac
done
```

```
#Criação de array para nomes de ficheiros
if [[ "$file_title" ]]; then
    if [[ ! -f $file_title ]]; then
        echo "[Erro] --> Ficheiro não encontrado!"
        $file_title="" #Reiniciar variável
    else
        array_ignore=($(create_array "$file_title")) #Criar array com nom
    fi
fi
```

   1.2. Mantenho a implementação usada no backup_files.sh para verificar a existência das diretorias necessárias.
      1.2.1. Adicionei uma verificação para o optarg de "-b", sendo passado um nome de ficheiro é verificada a sua existência e se existir este é passado como argumento para o script auxiliar **create_array.sh** que irá criar um array com o nome dos ficheiros/diretorias a ignorar (mais informações de como funciona em Funções/scripts auxiliares [17].

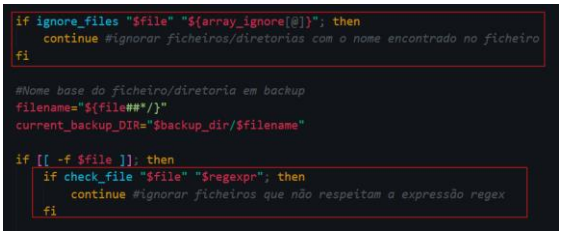
2. A parte principal do código neste script funciona da mesma forma que em backup_files.sh, mas neste caso tive de construir a parte do

```
if [[ -d $file ]]; then
        filename="${file##*/}"
        current_backup_DIR="$backup_dir/$filename" #path sub-diretoria

        if ignore_files "$file" "${array_ignore[@]}"; then
            continue #ignorar ficheiros/diretorias com o nome encontrado no ficheiro
        fi

        if [[ -e "$current_backup_DIR" ]]; then  #Verificar existência da sub-diretoria
            if [[ $Check_mode -eq 1 ]]; then
                echo "backup -c $file $current_backup_DIR"
                backup "$file" "$current_backup_DIR" #Função recursiva à sub-diretoria
            else
                log $log_file "backup "$file" "$current_backup_DIR""
                backup "$file" "$current_backup_DIR"
            fi
        else
            if [[ $Check_mode -eq 1 ]]; then
                echo "mkdir -p $current_backup_DIR"
                mkdir -p "$current_backup_DIR" || { echo "[ERRO] ao criar $current_backup_DIR"; ((counter_erro++)); continue;}   #Criar sub-
                echo "Sub-Diretoria $filename criada com sucesso!"
                backup "$file" "$current_backup_DIR"
            else
                mkdir -p "$current_backup_DIR" || { echo "[ERRO] ao criar $current_backup_DIR"; ((counter_erro++)); continue;}
                echo "Sub-Diretoria $filename criada com sucesso!"
                log $log_file "mkdir -p "$current_backup_DIR""
                log $log_file "backup "$dir" "$current_backup_DIR""
                backup "$file" "$current_backup_DIR"
            fi
```

código para resolver o problema de copiar também as subdiretorias presentes na diretoria fonte. Para isso defini uma função principal denominada backup que recebe como argumentos $1 e $2 o path para a subdiretoria fonte e a subdiretoria backup.

Para chamar a função recursivamente eu optei primeiro por uma resolução que utiliza um ciclo for onde iterava verificando se fosse ficheiro a fazer o mesmo que em backup_files.sh e se for uma subdiretoria a função vai chamar-se recursivamente, mas tendo como argumentos as subdiretorias fonte e backup.

2.1. Desenvolvimento das funções auxiliares para quando são passados os argumentos "-b" e "-r". Aqui defini duas funções ignore_files e check_file que são ativadas no início de cada iteração para verificar as condições.

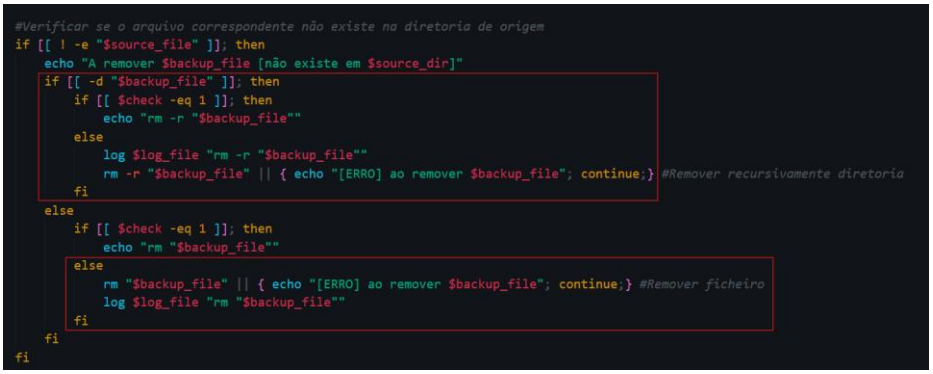


2.1.1. Na função **ignore_files** esta recebe o nome do ficheiro da iteração e o array criado pelo script create_array e itera comparando o nome do file com os nomes no array se encontrar correspondência retorna o e o ficheiro é ignorado na função principal. Se iterar por todo o array e não encontrar correspondência retorna 1 e não ignora o ficheiro. Função explicada em detalhe em Funções/scripts auxiliares [17];

2.1.2. Na função **check_file** esta recebe o nome do ficheiro da iteração e verifica se este não possui a expressão regex a partir do operador =~ se este não respeitar a expressão regex então será ignorado (retorna 0) e caso contrário será avaliado (retorna 1). Função explicada em detalhe em Funções/scripts auxiliares [17];

```
if [[ -d "$backup_file" ]]; then
    if [[ $check -eq 1 ]]; then
        echo "rm -r "$backup_file""
    else
        log $log_file "rm -r "$backup_file""
        rm -r "$backup_file" || { echo "[ERRO] ao remover $backup_file"; continue;}
    fi
```

3. A função **remove_files_NE** também está implementada, mas possui algumas alterações, pois para remover sub-diretorias recursivamente utilizamos o rm -r em vez de apenas rm. Por isso adicionei uma estrutura condicional. A função está explicada mais em detalhe em Funções/scripts auxiliares [17];

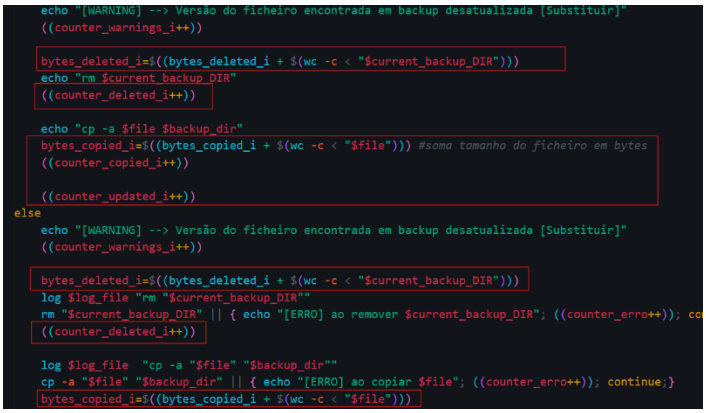


4. Implementações extra:

4.1. Também o funcionamento do script auxiliar **function_log.sh** está presente em Funções/scripts auxiliares [17]. Tem por objetivo guardar um ficheiro .log que possui informações sobre o backup.

3-Backup_summary.sh)

1. A ideia base deste script foi reutilizar o máximo de código construído em backup.sh adicionar contadores e fazer as alterações necessárias nas funções.



1.1. Variáveis que guardam os valores por subdiretoria e finais de contagem. Cada contador vai aumentar à medida que um warnig, erro, copia, remoção, update ocorre.

```
#Variáveis de contagem globais
counter_erro=0
counter_warnings=0
counter_copied=0
counter_deleted=0
counter_updated=0
bytes_deleted=0
bytes_copied=0

# Variáveis para contadores internos
counter_erro_i=0
counter_warnings_i=0
counter_copied_i=0
counter_deleted_i=0
counter_updated_i=0
bytes_deleted_i=0
bytes_copied_i=0
```

```
# Atualiza os contadores globais
counter_erro=$((counter_erro + counter_erro_i))
counter_warnings=$((counter_warnings + counter_warnings_i))
counter_updated=$((counter_updated + counter_updated_i))
counter_copied=$((counter_copied + counter_copied_i))
counter_deleted=$((counter_deleted + counter_deleted_i))
bytes_deleted=$((bytes_deleted + bytes_deleted_i))
bytes_copied=$((bytes_copied + bytes_copied_i))
```

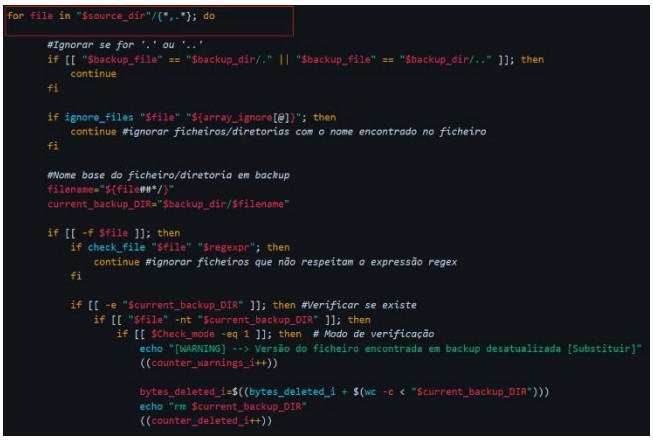
1.2. Também é guardado o tamanho de cada ficheiro através do comando wc -c (que retorna o tamanho do ficheiro em questão contando o tamanho das palavras, incluindo espaços brancos em bytes).

```
local counter_deleted_files=$(find "$backup_file" -type f | wc -l)
counter_deleted_i=$(($counter_deleted + $counter_deleted_files))
bytes_deleted_i=$(($bytes_deleted + $(du -sb "$backup_file" | cut -f1))) #Tamanho total da sub-diretoria | cut -f1 rema

log $log_file "rm -r "$backup_file""
rm -r "$backup_file" || { echo "[ERRO] ao remover $backup_file"; ((counter_erro++)); continue;} #Remover recursivamente
```

Na função remove_files_NE utilizei contadores, mas aqui tive de fazer umas pesquisas mais profundas para perceber como contar o tamanho total de uma subdiretoria que fosse removida recursivamente com o comando rm -r. Para isso utilizei o comando find para encontrar o número total de ficheiros apagados para o counter do número de ficheiros apagados e para a soma total de bytes utilizei o comando du (disk usage) que junto com o argumento -sb mostra o total de espaço ocupado pela subdiretoria em bytes. O comando cut ajuda para selecionar apenas a parte da

informação que necessito, neste caso ao usar o du -sb ele retorna o tamanho total em bytes e o nome da subdiretoria usando o cut com o argumento -f1 seleciono apenas o tamanho total que será somado ao counter de bytes removidos.

2. Alterações no modo como faço as iterações:
    2.1. Neste script para conseguir um summary sempre que acaba de copiar uma pasta usei dois ciclos for onde no primeiro vejo os ficheiros e no segundo itero pelas subdiretorias e executo recursivamente a função backup.



4-Backup_check.sh)

1. Para começar, defini bem o que é suposto este script fazer, que consiste em comparar duas diretorias passadas como argumentos e verificar se o seu conteúdo é igual ou não utilizando o comando md5sum. Caso sejam encontrados ficheiros diferentes, deve ser escrita uma mensagem idêntica a: "src/text.txt bak1/text.txt differ."

2. Sabendo agora o que é suposto o script fazer, fiz as verificações necessárias para que seja apenas passados como argumentos dois diretórios.

```
if [ "$#" -ne 2 ]; then
    echo "Erro! Deve apenas ter dois diretórios como argumentos!"
    exit 1
fi

#Verifica a existência da diretoria de origem
if [[ ! -d $1 ]]; then
    echo "Erro! O primeiro argumento não é um diretório!"
    exit 1
fi

#Verifica a existência da diretoria de origem
if [[ ! -d $2 ]]; then
    echo "Erro! O segundo argumento não é um diretório!"
    exit 1
fi
```

3. De seguida, defini como seria a estrutura do Código e decidi que para simplificar o Código, iria dividi-lo em duas funções, sendo elas "compare_files()" e "traverse_and_compare()", sendo que a segunda seria a "main" e chamaria a primeira dentro da mesma.

4. A função "compare_files()" seria chamada sempre que se quisesse comparar dois ficheiros, tendo-os como argumentos.

```
compare_files() {
    local src_dir="$1"
    local bkup_dir="$2"


    src_checksum=$(md5sum "$src_dir" | awk '{ print $1}')
    bkup_checksum=$(md5sum "$bkup_dir" | awk '{ print $1}')

    if [ "$src_checksum" != "$bkup_checksum" ]; then
        echo "$src_dir $bkup_dir differ"
    fi
}
```

   4.1 Esta função utilizaria o md5sum (e o awk) para obter o hashcode do ficheiro. Foi necessário a utilização do awk para guardar apenas os valores dos hashcodes nos respetivos checksums.

   4.2 De seguida, comparei os dois checksums e, caso fossem diferentes, o output seria a mensagem pretendida: "src/text.txt bak1/text.txt differ."

NOTA: A seguir vou utilizar o termo de níveis para explicar o Código. O que pretendo dizer com sub-diretórios e ficheiros um nível abaixo são aqueles que após o diretório principal têm apenas uma "/" no seu path. Isto é por exemplo "src_dir/subdir1" e "src_dir/ex.txt" estão um nível abaixo do diretório "src_dir" principal, mas "src_dir/subdir1/subdir11" já está dois níveis abaixo.

5. Finalmente, chegando à nossa função principal, “traverse_and_compare()”, guardei os nomes dos diretórios src e bkup passados como argumentos, na execução do script, nas variáveis locais “current_src_dir” e “current_bkup_dir” respetivamente. Utilizei “current”, porque vou chamar recursivamente esta função mais à frente.

```
traverse_and_compare() {
    local current_src_dir="$1"
    local current_bkup_dir="$2"

    for src_path in "$current_src_dir"/*; do
        relative_path="${src_path#$current_src_dir/}"
        relative_bkup_path="$current_bkup_dir/$relative_path"

        if [ -d "$src_path" ]; then
            if [ -d "$relative_bkup_path" ]; then
                traverse_and_compare "$src_path" "$relative_bkup_path"
            else
                echo "Erro! O subdiretório $relative_path não existe no $current_bkup_dir."
            fi

        elif [ -f "$src_path" ]; then
            if [ -f "$relative_bkup_path" ]; then
                compare_files "$src_path" "$relative_bkup_path"
            else
                echo "Erro! O ficheiro $relative_path não existe no $current_bkup_dir."
            fi
        fi
    done

}
```

6. De seguida, utilizando um ciclo for, iterei sobre todos os sub-diretórios e ficheiros um nível abaixo do diretório src principal (“src_path”).

7. Dentro desse ciclo for, criei as variáveis locais “relative _path” e “bkup_path”, sendo que o propósito do primeiro é apenas contruir o segundo.

    7.1 O meu objetivo era criar um path na diretoria bkup no mesmo nível que o “src_path” , ou seja criar um segundo path cuja única diferença do “src_path” é que antes da primeira “/” do path (não é a primeira quando é chamada recursivamente, mas acho que se percebe o que quero dizer) , o primeiro tenha o “current_src_dir” e o segundo tenha o “current_bkup_dir”.

8. Para criar esse segundo path (“bkup_path”), primeiro guardo no “relative _path” o caminho do “src_path” sem o “current_src_dir/”.

    8.1 Ou seja, “src_path” = “current_src_dir”/”relative _path”.
    8.2 Segundamente, crio o “bkup_path” = ”current_bkup_dir”/”relative _path”.

9. Depois, caso o “src_path” seja um diretório, verifico se a também existe a sua contraparte no bkup, o “bkup_path”.

    9.1 Caso não exista, dá um erro, pois o diretório com esse nome não existe no bkup.

    9.2 Caso exista, chamo recursivamente a função “traverse_and_compare()” com os subdiretórios (“src_path” e “bkup_path”) como argumentos que vai comparar os dois e repetir o processo todo até aqui.

10. Por outro lado, se o “src_path” for um ficheiro, volto a verificar se também existe a sua contraparte no bkup (de novo o “bkup_path”).

    10.1 Caso não exista dá erro, pois o ficheiro com esse nome não existe no bkup.
    10.2 Caso exista, chamo a função “compare_files()” com os dois ficheiros (“src_path” e “bkup_path”) como argumentos para os comparer e dar os devidos outputs.

11. Por fim, chamo a função “traverse_and_compare()” com os argumentos passados quando se executa o script (os diretórios src e bkup).

```
traverse_and_compare "$1" "$2"
```

PROBLEMA #1

12. Apercebi-me de um problema com o script, que é que caso o diretório backup tenha ficheiros extra, não dá nenhum erro. Isto acontece, pois, eu itero apenas sobre cada um dos ficheiros ou sub-diretórios dos diretórios src e verifico se existem no backup, mas não faço o contrário.

13. A solução que encontrei para o problema foi chamar a função “traverse_and_compare()” duas vezes uma vez com $1=src e $2=bkup e outra com $1=bkup e $2=src.

```
traverse_and_compare "$1" "$2"

traverse_and_compare "$2" "$1"
```

PROBLEMA #2

14. Fazê-lo desta maneira resolvia o problema de não contar como erro o diretório bkup ter ficheiros extra, mas causava outro: cada vez que se verificava se dois ficheiros são iguais (“compare_files()”) os outputs eram a dobrar, ficaria algo do género:

    “src/text.txt bak1/text.txt differ.”

    “bak1/text.txt bak1/src.txt differ.”

    Ou seja, fazia a mesma coisa duas vezes.

15. A maneira que encontrei de solucionar este problema foi criar uma variável global chamada “sec_run” que, caso fosse diferente de 0 durante a execução do “traverse_and_compare()”, impossibilita a chamada da função “compare_files”, desta maneira não haveria outputs a dobrar.

```
elif [ -f "$src_path" ]; then
    if [ -f "$relative_bkup_path" ]; then
        if [ "$sec_run" -eq 0 ]; then
            compare_files "$src_path" "$relative_bkup_path"
        fi
```

```
traverse_and_compare() {
    local current_src_dir="$1"
    local current_bkup_dir="$2"

    for src_path in "$current_src_dir"/*; do
        relative_path="${src_path#$current_src_dir/}"
        relative_bkup_path="$current_bkup_dir/$relative_path"

        if [ -d "$src_path" ]; then
            if [ -d "$relative_bkup_path" ]; then
                traverse_and_compare "$src_path" "$relative_bkup_path"
            else
                echo "Erro! O subdiretório $relative_path não existe no $current_bkup_dir."
            fi

        elif [ -f "$src_path" ]; then
            if [ -f "$relative_bkup_path" ]; then
                if [ "$sec_run" -eq 0 ]; then
                    compare_files "$src_path" "$relative_bkup_path"
                fi
            else
                echo "Erro! O ficheiro $relative_path não existe no $current_bkup_dir."
            fi
        fi
    done
}
```

16. Assim, se fosse 0 na primeira e 1 na segunda execução do “traverse_and_compare()” estaria apenas a verificar se não existe nenhum ficheiro ou sub-diretório extra no diretório bkup.

```
sec_run=0

traverse_and_compare "$1" "$2"

sec_run=1

traverse_and_compare "$2" "$1"
```

FUNCIONALIDADE EXTRA #1

17. Por fim, decidi adicionar duas variáveis globais extra para contar o número de erros e o número de ficheiros extra, “count_diff” e “count_eq” respetivamente.

```
count_diff=0
count_eq=0
sec_run=0
```

18. Inicializei-as a 0s e fiz a respetiva contagem de erros e ficheiros iguais nestas partes do Código:

```
compare_files() {
    local src_dir="$1"
    local bkup_dir="$2"


    src_checksum=$(md5sum "$src_dir" | awk '{ print $1}')
    bkup_checksum=$(md5sum "$bkup_dir" | awk '{ print $1}')

    if [ "$src_checksum" != "$bkup_checksum" ]; then
        echo "$src_dir $bkup_dir differ"
        ((count_diff++))
    else
        ((count_eq++))
    fi
}
```

```
traverse_and_compare() {
    local current_src_dir="$1"
    local current_bkup_dir="$2"

    for src_path in "$current_src_dir"/*; do
        relative_path="${src_path#$current_src_dir/}"
        relative_bkup_path="$current_bkup_dir/$relative_path"

        if [ -d "$src_path" ]; then
            if [ -d "$relative_bkup_path" ]; then
                traverse_and_compare "$src_path" "$relative_bkup_path"
            else
                ((count_diff++))
                echo "Erro! O subdiretório $relative_path não existe no $current_bkup_dir."
            fi

        elif [ -f "$src_path" ]; then
            if [ -f "$relative_bkup_path" ]; then
                if [ "$sec_run" -eq 0 ]; then
                    compare_files "$src_path" "$relative_bkup_path"
                fi
            else
                ((count_diff++))
                echo "Erro! O ficheiro $relative_path não existe no $current_bkup_dir."
            fi
        fi
    done
}
```

19. A seguir, gerei um output que revelasse o número de erros (diferenças entre os diretórios) e o número de ficheiros iguais.

```
echo "Número de erros total: $count_diff"
echo "Número de ficheiros iguais: $count_eq"
```

FUNCIONALIDADES EXTRA #2

20. Por fim, adicionei um output a informar se os dois diretórios são iguais ou não.

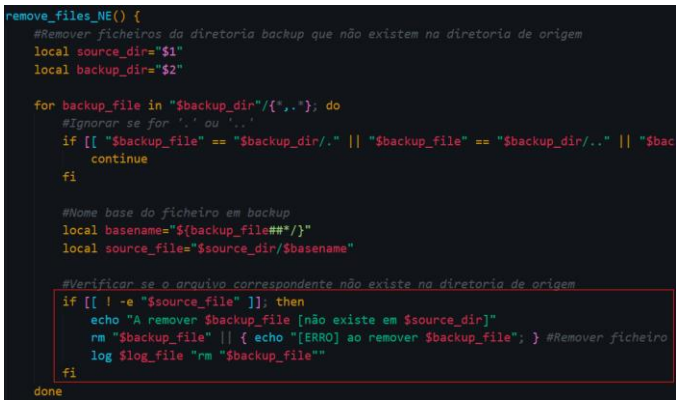
```
if [ "$count_diff" -eq 0 ]; then
    echo "Os diretórios são iguais"
else
    echo "Os diretórios são diferentes"
fi
```

Funções/Scripts auxiliares

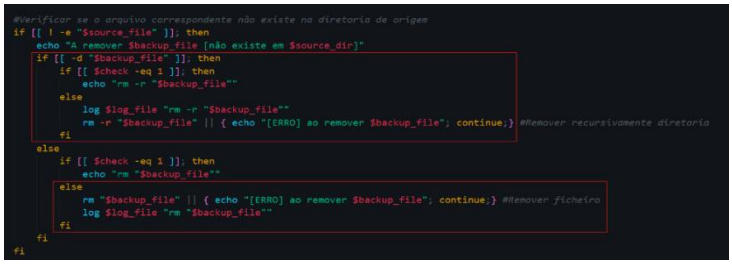
- **Função remove_files_NE**

  Esta função recebe como argumentos o path da diretoria fonte e da diretoria backup e ainda a variável check, para fazer apenas echo dos comandos que seriam executados nesta função. A função tem pequenas diferenças de script para script:

  - Backup_files.sh: Esta função é mais simples, pois só remove ficheiros da diretoria backup que não estão presentes na fonte. Tive alguns problemas com a pattern usada na iteração /{*,.*}, pois ao iterar por todo o tipo de ficheiros também iterava pelos ficheiros ocultos que referenciam a diretoria atual e a diretoria pai. Tive outro problema nesta iteração relacionada com um erro que está explicado na secção Debugging;



  - Backup.sh: Aqui a alteração foi lidar com a remoção recursiva de subdiretorias inteiras da backup com o argumento -r (recursive) no comando rm. Verificamos se é subdiretoria e depois removemos ou apenas aparece o echo correspondente, dependendo do check mode;



  - Backup_summary.sh: As alterações realizadas a esta função neste script em relação à função utilizada no backup.sh baseiam-se apenas em contadores que já estão explicados na secção do script backup_summary.sh.

  - Problema em remove_files_NE, ao iterar por todos os ficheiros, quando analisava uma pasta sem ficheiros ou com dava erro ou removia tudo indevidamente . Depois de pesquisar percebi que bastava ignore essas duas patterns.

```
A remover ../backup/pasta2/.* [não existe em ../testes/pasta2]
rm: impossível remover '../backup/pasta2/.*': Ficheiro ou pasta inexistente
[ERRO] ao remover ../backup/pasta2/.*
```

```
|| "$backup_file" == "$backup_dir/.*" || "$backup_file" == "$backup_dir/*" ]]; then
```

- **Script create_array.sh**

Possui uma função com o mesmo nome, decidi criar um script para implementar mais dos conhecimentos que adquirimos em sistemas operativos em relação a comandos bash, usando source
carreguei o script create_array nos scripts principais, sendo assim possível utilizar as funções do mesmo.

```
source ./create_array.sh
```

```bash
#!/bin/bash

#Função auxiliar que converte lista de nomes de fixheiro .txt em array de nomes

create_array() {
    #Define array para guardar nomes de ficheiros a ser ignorados
    local name_files=()

    #Lógica para retirar nomes do ficheiro e colocar no array
    while IFS= read -r line; do
        if [ -z "$line" ]; then #Verifica se está vazio (se estiver ignora)
            continue
        fi
        name_files+=("$line") #Adicionar linha ao array
    done < "$1"

    echo "${name_files[@]}" #Retorna array
    return 0
}
```

A função é simples, tirei ideias dos scripts feitos nas aulas práticas. Crio um array local e num ciclo while vou lendo as linhas do ficheiro com o comando read -r line e adicionando-as ao array, o ciclo acaba quando o read chegar ao fim do ficheiro "done < $1". No fim retorno o array completo com um echo que será recebido numa variável no script principal.

- **Função ignore_files**

Esta função funciona exatamente da mesma forma em todos os scripts principais. Na receção dos argumentos tive alguns problemas relacionados com o array, pois sem o shift que adicionei entre as duas variáveis locais a variável que deveria ter o array de ficheiros a remover também recebia no array o primeiro argumento e então ignorava todos os ficheiros pois o nome do ficheiro estaria sempre no array.

```
#Argumentos necessários
local file_ig="$1"
shift
local array_ignore=("$@")
```

O file_ig deve ter o realpath, caso os nomes no ficheiro.txt tenham formas do tipo "../ou ../path/" .

```
ignore_files() {
    #Função auxiliar para verificar se nome de determinado
    #ficheiro se encontra no array de nomes de ficheiros a ignorar.

    #Argumentos necessários
    local file_ig=$(realpath "$1")
    dirpath="${file_ig%/*}"
    shift
    local array_ignore=("$@")
    for f in "${array_ignore[@]}"; do
        basename="${f##*/}"
        if [[ -e "$file_ig" ]]; then
            if [[ "$file_ig" == "$dirpath/$basename" ]]; then
                return 0 #Ficheiro ignorado
            fi
        fi
    done
```

Para iterar pelo array todo de nomes utilizei a expressão [@] que se refere a todos os argumentos presentes no array, faz a comparação do nome e retorna 0 se for igual e 1 se não for.

- **Função check_file**

  Esta função também permanece igual nos scripts principais em que é usada. Recebe os dois argumentos o nome de ficheiro e expressão regex que são comparados verificando se a expressão regex não é vazia com "-n" e se o basename (nome do ficheiro sem o path, conseguido usando a expressão ${file_a##*/}), se não respeitar o regex não será removido (return 0) e se respeitar o regex então poderá ser copiado.

```
check_file() {
    local file_a="$1"
    local regexpr="$2"

    local basename="${file_a##*/}"

    if [[ -n "$regexpr" && ! "$basename" =~ $regexpr ]]; then
        return 0  #Arquivo não respeita regex não será copiado
    fi

    return 1 #Arquivo vai ser copiado, pois respeita regex
}
```

- **Script function_log.sh (Função extra adicionada por mim)**

  Para o script ter uma função extra, decidi adicionar esta função depois de ter utilizado uma função parecida na aula prática nº6.

  Possui uma única função de nome log, a função é bastante simples, recebe um ficheiro .log e o command que está a ser executado e, usando o date guarda a hora atual, e escreve-o no ficheiro a partir do comando echo. Nos scripts principais é criado o ficheiro .log com a data atual e o nome do script e isto apenas ocorre quando o argumento de check "-c" não está ativo.

```
#Função que cria LOG do backup
log() {

    #Comando executado
    local log_file="$1"
    local command="$2"

    time_LOG=$(date +"%H:%M:%S")
    #Regista o comando no arquivo log
    echo "[$time_LOG] - $command" >> "$log_file"
}
```

```
if [[ $Check_mode -eq 0 ]]; then #Se check mode ativo não irá fazer log
    #Log file
    ##Obtém o data + horário atual
    time_LOG=$(date +"%H:%M:%S")
    LOG_date=$(date +"%d_%B_%Y")
    log_file="Backup[$LOG_date"-"$time_LOG].log"
    touch $log_file

    echo "|Log backup da diretoria $Source_DIR |" >> $log_file
    echo "-----------------------------------------------------------------------------------------------" >> $log_file
fi
```

```
#Criação de array para nomes de ficheiros
if [[ "$file_title" ]]; then
    if ! [[ -f $file_title ]]; then
        echo "[Erro] --> Ficheiro não encontrado!"
        $file_title="" #Reiniciar variável
    else
        array_ignore=($(create_array "$file_title"))
    fi
fi
```

Testes Realizados

Testes de backup_files.sh: Aqui utilizei uma diretoria com alguns ficheiros aleatórios (incluindo escondidos) e não possuía a diretoria destino pre-criada.

```
paul-pc@paulo-HP-LINUX:~/UA/SO/Projeto1-SO-$ ./backup_files.sh -c ../testes ../backup
mkdir -p ../backup
cp -a ../testes/f12.sh ../backup
cp -a ../testes/f1.txt ../backup
cp -a ../testes/f30.sh ../backup
cp -a ../testes/f60.sh ../backup
cp -a ../testes/.hidden ../backup
paul-pc@paulo-HP-LINUX:~/UA/SO/Projeto1-SO-$
```

Testes de backup.sh: Testes realizados com diretoria possuindo subdiretorias (escondidas também), assim como ficheiros e utilização de ficheiro de texto file_ig.txt (para testar parâmetro -b)

- Tentativa de usar recursividade de script: Tentei numa primeira abordagem usar a recursividade do próprio script. Mas tive problemas com ciclos infinitos.

```
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
[WARNING] --> Sem ficheiro atribuido!erro
```

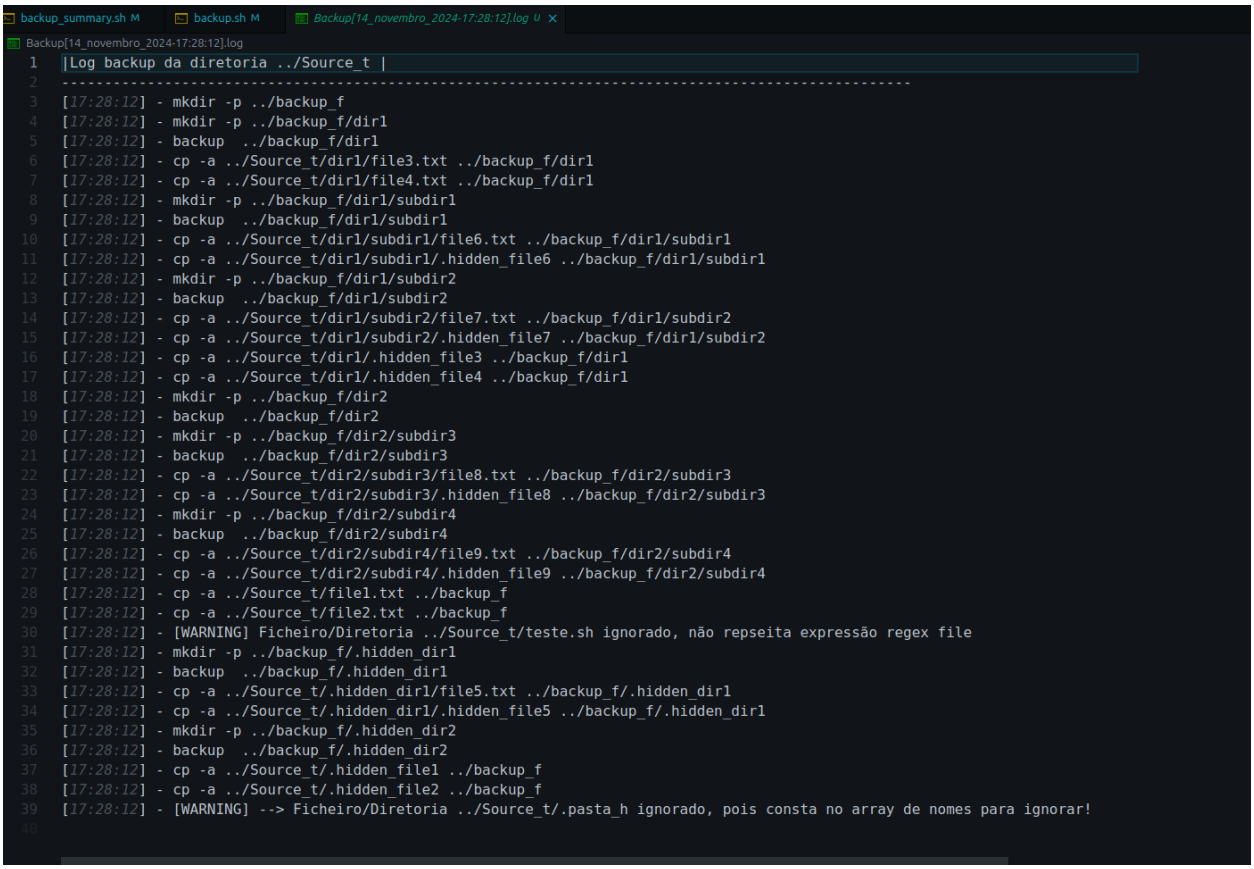
- Recursividade utilizando uma função dentro do script resolveu o problema e mostrou-se menos problemático.

```
paul-pc@paulo-HP-LINUX:~/UA/SO/Projeto1-SO$ ./backup.sh -b ../file_ig.txt -r file ../Source_t ../backup_f
Backup[14_novembro_2024-17:28:12] Sub-Diretoria dir1 criada com sucesso!
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/dir1/file3.txt copiado para ../backup_f/dir1]
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/dir1/file4.txt copiado para ../backup_f/dir1]
Backup[14_novembro_2024-17:28:12] Sub-Diretoria subdir1 criada com sucesso!
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/dir1/subdir1/file6.txt copiado para ../backup_f/dir1/subdir1]
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/dir1/subdir1/.hidden_file6 copiado para ../backup_f/dir1/subdir1]
Backup[14_novembro_2024-17:28:12] Sub-Diretoria subdir2 criada com sucesso!
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/dir1/subdir2/file7.txt copiado para ../backup_f/dir1/subdir2]
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/dir1/subdir2/.hidden_file7 copiado para ../backup_f/dir1/subdir2]
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/dir1/.hidden_file3 copiado para ../backup_f/dir1]
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/dir1/.hidden_file4 copiado para ../backup_f/dir1]
Backup[14_novembro_2024-17:28:12] Sub-Diretoria dir2 criada com sucesso!
Backup[14_novembro_2024-17:28:12] Sub-Diretoria subdir3 criada com sucesso!
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/dir2/subdir3/file8.txt copiado para ../backup_f/dir2/subdir3]
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/dir2/subdir3/.hidden_file8 copiado para ../backup_f/dir2/subdir3]
Backup[14_novembro_2024-17:28:12] Sub-Diretoria subdir4 criada com sucesso!
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/dir2/subdir4/file9.txt copiado para ../backup_f/dir2/subdir4]
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/dir2/subdir4/.hidden_file9 copiado para ../backup_f/dir2/subdir4]
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/file1.txt copiado para ../backup_f]
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/file2.txt copiado para ../backup_f]
[WARNING] --> Ficheiro ../Source_t/teste.sh ignorado, não repseita expressão regex file
Backup[14_novembro_2024-17:28:12] Sub-Diretoria .hidden_dir1 criada com sucesso!
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/.hidden_dir1/file5.txt copiado para ../backup_f/.hidden_dir1]
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/.hidden_dir1/.hidden_file5 copiado para ../backup_f/.hidden_dir1]
Backup[14_novembro_2024-17:28:12] Sub-Diretoria .hidden_dir2 criada com sucesso!
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/.hidden_file1 copiado para ../backup_f]
Backup[14_novembro_2024-17:28:12] [Ficheiro ../Source_t/.hidden_file2 copiado para ../backup_f]
[WARNING] --> Ficheiro/Diretoria ../Source_t/.pasta_h ignorado, pois consta no array de nomes para ignorar!
```

Testes para backup_summary.sh: Aqui utilizei exatamente a mesma diretoria fonte que em backup.sh

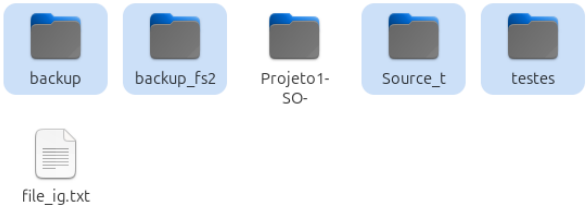
```
paul-pc@paulo-HP-LINUX:~/UA/SO/Projeto1-SO$ ./backup_summary.sh -b ../file_ig.txt -r file ../Source_t ../backup_fs
Backup[14_novembro_2024-17:30:06] [Ficheiro ../Source_t/file1.txt copiado para ../backup_fs]
Backup[14_novembro_2024-17:30:06] [Ficheiro ../Source_t/file2.txt copiado para ../backup_fs]
[WARNING] --> Ficheiro ../Source_t/teste.sh ignorado, não repseita expressão regex file
Backup[14_novembro_2024-17:30:06] [Ficheiro ../Source_t/.hidden_file1 copiado para ../backup_fs]
Backup[14_novembro_2024-17:30:06] [Ficheiro ../Source_t/.hidden_file2 copiado para ../backup_fs]
[WARNING] --> Ficheiro ../Source_t/.pasta_h ignorado, pois consta no array de nomes para ignorar!
While backuping files of ../Source_t: 0 Errors; 2 Warnings; 0 Updated; 4 Copied (124 B); 0 Deleted (0 B)
-------------------------------------------------
Backup[14_novembro_2024-17:30:06] Sub-Diretoria dir1 criada com sucesso!
Backup[14_novembro_2024-17:30:06] [Ficheiro ../Source_t/dir1/file3.txt copiado para ../backup_fs/dir1]
Backup[14_novembro_2024-17:30:06] [Ficheiro ../Source_t/dir1/file4.txt copiado para ../backup_fs/dir1]
Backup[14_novembro_2024-17:30:06] [Ficheiro ../Source_t/dir1/.hidden_file3 copiado para ../backup_fs/dir1]
Backup[14_novembro_2024-17:30:06] [Ficheiro ../Source_t/dir1/.hidden_file4 copiado para ../backup_fs/dir1]
While backuping files of ../Source_t/dir1: 0 Errors; 0 Warnings; 0 Updated; 4 Copied (144 B); 0 Deleted (0 B)
-------------------------------------------------
Backup[14_novembro_2024-17:30:06] Sub-Diretoria subdir1 criada com sucesso!
Backup[14_novembro_2024-17:30:06] [Ficheiro ../Source_t/dir1/subdir1/file6.txt copiado para ../backup_fs/dir1/subdir1]
Backup[14_novembro_2024-17:30:06] [Ficheiro ../Source_t/dir1/subdir1/.hidden_file6 copiado para ../backup_fs/dir1/subdir1]
While backuping files of ../Source_t/dir1/subdir1: 0 Errors; 0 Warnings; 0 Updated; 2 Copied (88 B); 0 Deleted (0 B)
-------------------------------------------------
Backup[14_novembro_2024-17:30:06] Sub-Diretoria subdir2 criada com sucesso!
Backup[14_novembro_2024-17:30:06] [Ficheiro ../Source_t/dir1/subdir2/file7.txt copiado para ../backup_fs/dir1/subdir2]
Backup[14_novembro_2024-17:30:06] [Ficheiro ../Source_t/dir1/subdir2/.hidden_file7 copiado para ../backup_fs/dir1/subdir2]
While backuping files of ../Source_t/dir1/subdir2: 0 Errors; 0 Warnings; 0 Updated; 2 Copied (88 B); 0 Deleted (0 B)
-------------------------------------------------
Backup[14_novembro_2024-17:30:06] Sub-Diretoria dir2 criada com sucesso!
While backuping files of ../Source_t/dir2: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0 B); 0 Deleted (0 B)
-------------------------------------------------
Backup[14_novembro_2024-17:30:06] Sub-Diretoria subdir3 criada com sucesso!
Backup[14_novembro_2024-17:30:06] [Ficheiro ../Source_t/dir2/subdir3/file8.txt copiado para ../backup_fs/dir2/subdir3]
Backup[14_novembro_2024-17:30:06] [Ficheiro ../Source_t/dir2/subdir3/.hidden_file8 copiado para ../backup_fs/dir2/subdir3]
While backuping files of ../Source_t/dir2/subdir3: 0 Errors; 0 Warnings; 0 Updated; 2 Copied (88 B); 0 Deleted (0 B)
-------------------------------------------------
Backup[14_novembro_2024-17:30:06] Sub-Diretoria subdir4 criada com sucesso!
Backup[14_novembro_2024-17:30:06] [Ficheiro ../Source_t/dir2/subdir4/file9.txt copiado para ../backup_fs/dir2/subdir4]
Backup[14_novembro_2024-17:30:06] [Ficheiro ../Source_t/dir2/subdir4/.hidden_file9 copiado para ../backup_fs/dir2/subdir4]
While backuping files of ../Source_t/dir2/subdir4: 0 Errors; 0 Warnings; 0 Updated; 2 Copied (88 B); 0 Deleted (0 B)
-------------------------------------------------
Backup[14_novembro_2024-17:30:06]:
Backup Summary: 0 Errors; 2 Warnings; 0 Updated; 16 Copied (620 B); 0 Deleted (0 B)
-------------------------------------------------
```

Exemplo de um ficheiro.log



```
backup_summary.sh M   backup.sh M   Backup[14_novembro_2024-17:28:12].log U
Backup[14_novembro_2024-17:28:12].log
1  |Log backup da diretoria ../Source_t |
2  ------------------------------------------------------------------------------------------
3  [17:28:12] - mkdir -p ../backup_f
4  [17:28:12] - mkdir -p ../backup_f/dir1
5  [17:28:12] - backup  ../backup_f/dir1
6  [17:28:12] - cp -a ../Source_t/dir1/file3.txt ../backup_f/dir1
7  [17:28:12] - cp -a ../Source_t/dir1/file4.txt ../backup_f/dir1
8  [17:28:12] - mkdir -p ../backup_f/dir1/subdir1
9  [17:28:12] - backup  ../backup_f/dir1/subdir1
10 [17:28:12] - cp -a ../Source_t/dir1/subdir1/file6.txt ../backup_f/dir1/subdir1
11 [17:28:12] - cp -a ../Source_t/dir1/subdir1/.hidden_file6 ../backup_f/dir1/subdir1
12 [17:28:12] - mkdir -p ../backup_f/dir1/subdir2
13 [17:28:12] - backup  ../backup_f/dir1/subdir2
14 [17:28:12] - cp -a ../Source_t/dir1/subdir2/file7.txt ../backup_f/dir1/subdir2
15 [17:28:12] - cp -a ../Source_t/dir1/subdir2/.hidden_file7 ../backup_f/dir1/subdir2
16 [17:28:12] - cp -a ../Source_t/dir1/.hidden_file3 ../backup_f/dir1
17 [17:28:12] - cp -a ../Source_t/dir1/.hidden_file4 ../backup_f/dir1
18 [17:28:12] - mkdir -p ../backup_f/dir2
19 [17:28:12] - backup  ../backup_f/dir2
20 [17:28:12] - mkdir -p ../backup_f/dir2/subdir3
21 [17:28:12] - backup  ../backup_f/dir2/subdir3
22 [17:28:12] - cp -a ../Source_t/dir2/subdir3/file8.txt ../backup_f/dir2/subdir3
23 [17:28:12] - cp -a ../Source_t/dir2/subdir3/.hidden_file8 ../backup_f/dir2/subdir3
24 [17:28:12] - mkdir -p ../backup_f/dir2/subdir4
25 [17:28:12] - backup  ../backup_f/dir2/subdir4
26 [17:28:12] - cp -a ../Source_t/dir2/subdir4/file9.txt ../backup_f/dir2/subdir4
27 [17:28:12] - cp -a ../Source_t/dir2/subdir4/.hidden_file9 ../backup_f/dir2/subdir4
28 [17:28:12] - cp -a ../Source_t/file1.txt ../backup_f
29 [17:28:12] - cp -a ../Source_t/file2.txt ../backup_f
30 [17:28:12] - [WARNING] Ficheiro/Diretoria ../Source_t/teste.sh ignorado, não repseita expressão regex file
31 [17:28:12] - mkdir -p ../backup_f/.hidden_dir1
32 [17:28:12] - backup  ../backup_f/.hidden_dir1
33 [17:28:12] - cp -a ../Source_t/.hidden_dir1/file5.txt ../backup_f/.hidden_dir1
34 [17:28:12] - cp -a ../Source_t/.hidden_dir1/.hidden_file5 ../backup_f/.hidden_dir1
35 [17:28:12] - mkdir -p ../backup_f/.hidden_dir2
36 [17:28:12] - backup  ../backup_f/.hidden_dir2
37 [17:28:12] - cp -a ../Source_t/.hidden_file1 ../backup_f
38 [17:28:12] - cp -a ../Source_t/.hidden_file2 ../backup_f
39 [17:28:12] - [WARNING] --> Ficheiro/Diretoria ../Source_t/.pasta_h ignorado, pois consta no array de nomes para ignorar!
40
```

Diretorias e ficheiros utilizados para teste:



- Um problema que encontrei durante a revisão do código em todos os scripts foi redundâncias relacionadas à má definição das condições, pois ao definir primeiro a condição de check tive de escrever duas vezes o mesmo código para a condição de file e diretoria, etc. Incluído no Apêndice [26].

- Modificação final de alguns echo para ficarem todos iguais, utilização de expressões, tais como: %.* (remove extensão ".xxx" do ficheiro) e ##*/ (remove "path/" de ficheiro ou diretoria);

- Reorganização de algumas estruturas de condição, para não serem executadas desnecessariamente. Por exemplo se a diretoria fonte não existir o programa acaba, logo não faz sentido criar arrays e ficheiros .log antes dessas verificações iniciais.

- Para que o teste enviado pelo professor (testado no backup_summary.sh) dê o resultado pretendido (em termos de output), comentando as linhas: 225, 226, 247, 248, 257, 305, 335-7;

Conclusão

Com o desenvolvimento deste trabalho conseguimos ficar a entender melhor o funcionamento de comandos bash, incluindo o funcionamento de argumentos e de inputs de argumentos, o modo de funcionamento de arrays e as suas propriedades, interação entre scripts, desenvolvimento de funções, retorno de funções, etc.

Durante o desenvolvimento deste trabalho conseguimos aplicar muito do conhecimento que nos foi transmitido nas aulas de sistemas operativos, conseguindo assim desenvolver e perceber o funcionamento de todos os comandos bash que utilizados para o fim do bom funcionamento dos scripts. Conseguimos ainda tratar todos os erros encontrados durante a fase de testes e a partir disso construir um código mais sólido e funcional.

Os scripts desenvolvidos cumprem com os seus objetivos e demonstram como o desenvolvimento de scripts Bash é uma mais-valia no Linux para simplificar tarefas e facilitar a vida de quem trabalha com o mesmo.

Todo o desenvolvimento e histórico do código pode ser consultado na página de git hub:
https://github.com/Paul-Y5/Projeto1-SO-

Web Grafia

- https://stackabuse.com/get-total-size-of-a-directory-in-linux/
- https://www.howtogeek.com/766978/how-to-use-case-statements-in-bash-scripts/
- https://www.computerhope.com/unix/bash/shift.htm
- https://blog.tratif.com/2023/01/09/bash-tips-1-logging-in-shell-scripts/
- https://www.linuxforce.com.br/comandos-linux/comandos-linux-comando-getopts/
- https://guialinux.uniriotec.br/
- https://www.linuxjournal.com/content/pattern-matching-bash

Ideias retiradas de aulas práticas de SO: Arrays; funções; log; iterações(for); boas práticas;

Apêndice

- **Antes:**

```
    filename="${file##*/}"
    current_backup_DIR="$backup_dir/$filename"

    if [[ -f $file ]]; then
        if [[ $Check_mode -eq 1 ]]; then  # Modo de verificação
            if [[ -e "$current_backup_DIR" ]]; then
                #Remover ficheiros que não existem na source
                if [[ "$file" -nt "$current_backup_DIR" ]]; then
                    echo "[WARNING] --> Versão do ficheiro encontrada em backup desatualizada [Substituir]"

                    echo "rm $current_backup_DIR"

                    echo "cp -a $file $backup_dir"
                else
                    echo "[WARNING] --> Backup possui versão mais recente do ficheiro $file --> [Não copiado]"
                fi
            else
                echo "cp -a $file $backup_dIR"
            fi
        else
            if [[ -e "$current_backup_DIR" ]]; then
                if [[ "$file" -nt "$current_backup_DIR" ]]; then
                    echo "[WARNING] --> Versão do ficheiro encontrada em backup desatualizada [Substituir]"

                    log $log_file "rm "$current_backup_DIR""
                    rm "$current_backup_DIR" || { echo "[ERRO] ao remover $current_backup_DIR"; continue;}

                    log $log_file  "cp -a "$file" "$backup_dir""
                    cp -a "$file" "$backup_dir" || { echo "[ERRO] ao copiar $file"; continue;}

                    log $log_file "[Warning --> Substituído]"
                else
                    log $log_file "[Warning --> Não substituído]"
                    echo "[WARNING] --> Backup possui versão mais recente do ficheiro $file --> [Não substituido]"
                fi
            else
                echo "[Ficheiro $file copiado para backup]"
                log $log_file "cp -a "$file" "$backup_dir""
                cp -a "$file" "$backup_dir" || { echo "[ERRO] ao copiar $file"; continue;}
            fi
        fi
    fi
done

# Imprime o status após processar arquivos
echo "While backuping files of $source_dir: $counter_erro_i Errors; $counter_warnings_i Warnings; $counter_updated_i Updated; $counter_copied_i Copied ($bytes_copied_i B); $counter_deleted_i Deleted ($bytes_deleted_i B)"
echo "------------------------------------------------"

for dir in "$source_dir"/{*,.*}; do
    if [[ -d $dir ]]; then
        filename="${dir##*/}"
        current_backup_DIR="$backup_dir/$filename"

        if ignore_files "$file" "${array_ignore[@]}"; then
            continue #ignorar ficheiros/diretorias com o nome encontrado no ficheiro
        fi

        if [[ $Check_mode -eq 1 ]]; then
            if [[ -e "$current_backup_DIR" ]]; then  #Verificar existência da sub-diretoria
                echo "backup -c $dir $current_backup_DIR"
                backup "$dir" "$current_backup_DIR" #Função recursiva à sub-diretoria
            else
                echo "mkdir -p $current_backup_DIR"
                mkdir -p "$current_backup_DIR" || { echo "[ERRO] ao criar $current_backup_DIR"; ((counter_erro++)); continue;}   #criar sub-diretoria
                echo "Sub-Diretoria $filename criada com sucesso!"
                backup "$dir" "$current_backup_DIR"
            fi
        else
            if [[ -e "$current_backup_DIR" ]]; then
                log $log_file "backup "$dir" "$current_backup_DIR""
                backup "$dir" "$current_backup_DIR"
            else
                mkdir -p "$current_backup_DIR" || { echo "[ERRO] ao criar $current_backup_DIR"; ((counter_erro++)); continue;}
                echo "Sub-Diretoria $filename criada com sucesso!"
                log $log_file "mkdir -p "$current_backup_DIR""
                log $log_file "backup "$dir" "$current_backup_DIR""
                backup "$dir" "$current_backup_DIR"
            fi
        fi
    fi
done

return 0
```

➢ **Depois:**

```
for file in "$source_dir"/{*,.*}; do

        #Ignorar se for '.' ou '..'
        if [[ "$backup_file" == "$backup_dir/." || "$backup_file" == "$backup_dir/.." ]]; then
            continue
        fi

        if ignore_files "$file" "${array_ignore[@]}"; then
            continue #Ignorar ficheiros/diretorias com o nome encontrado no ficheiro
        fi

        #Nome base do ficheiro/diretoria em backup
        filename="${file##*/}"
        current_backup_DIR="$backup_dir/$filename"

        if [[ -f $file ]]; then
            if check_file "$file" "$regexpr"; then
                continue #Ignorar ficheiros que não respeitam a expressão regex
            fi

            if [[ -e "$current_backup_DIR" ]]; then #Verificar se existe
                if [[ "$file" -nt "$current_backup_DIR" ]]; then
                    if [[ $Check_mode -eq 1 ]]; then  # Modo de verificação
                        echo "[WARNTNG] --> Versão do ficheiro encontrada em backup desatualizada [Substituir]"
                        ((counter_warnings_i++))

                        bytes_deleted_i=$((bytes_deleted_i + $(wc -c < "$current_backup_DIR")))
                        echo "rm $current_backup_DIR"
                        ((counter_deleted_i++))

                        echo "cp -a $file $backup_dir"
                        bytes_copied_i=$((bytes_copied_i + $(wc -c < "$file"))) #soma tamanho do ficheiro em bytes
                        ((counter_copied_i++))

                        ((counter_updated_i++))
                    else
                        echo "[WARNTNG] --> Versão do ficheiro encontrada em backup desatualizada [Substituir]"
                        ((counter_warnings_i++))

                        bytes_deleted_i=$((bytes_deleted_i + $(wc -c < "$current_backup_DIR")))
                        log $log_file "rm "$current_backup_DIR""
                        rm "$current_backup_DIR" || { echo "[ERRO] ao remover $current_backup_DIR"; ((counter_erro++)); continue;}
                        ((counter_deleted_i++))

                        log $log_file  "cp -a "$file" "$backup_dir""
                        cp -a "$file" "$backup_dir" || { echo "[ERRO] ao copiar $file"; ((counter_erro++)); continue;}
                        bytes_copied_i=$((bytes_copied_i + $(wc -c < "$file")))
                        ((counter_copied_i++))

                        log $log_file "[Warning --> Substituído]"
                        ((counter_updated_i++))
                    fi
                else
                    if [[ $Check_mode -eq 1 ]]; then  # Modo de verificação
                        echo "[WARNTNG] --> Backup possui versão mais recente do ficheiro $file --> [Não copiado]"
                        ((counter_warnings_i++))
                    else
                        log $log_file "[Warning --> Não substituído]"
                        echo "[WARNTNG] --> Backup possui versão mais recente do ficheiro $file --> [Não substituído]"
                        ((counter_warnings_i++))
                    fi
                fi
            else
                if [[ $Check_mode -eq 1 ]]; then  # Modo de verificação
                    echo "cp -a $file $backup_dir"
                    ((counter_copied_i++))
                    bytes_copied_i=$((bytes_copied_i + $(wc -c < "$file")))
                else
                    echo "|Ficheiro $file copiado para backup|"
                    log $log_file "cp -a "$file" "$backup_dir""
                    cp -a "$file" "$backup_dir" || { echo "[ERRO] ao copiar $file"; ((counter_erro++)); continue;}
                    ((counter_copied_i++))
                    bytes_copied_i=$((bytes_copied_i + $(wc -c < "$file")))
                fi
            fi
        fi
    done

    # Atualiza os contadores globais
    counter_erro=$((counter_erro + counter_erro_i))
    counter_warnings=$((counter_warnings + counter_warnings_i))
    counter_updated=$((counter_updated + counter_updated_i))
    counter_copied=$((counter_copied + counter_copied_i))
    counter_deleted=$((counter_deleted + counter_deleted_i))
    bytes_deleted=$((bytes_deleted + bytes_deleted_i))
    bytes_copied=$((bytes_copied + bytes_copied_i))

    # Imprime o status após processar arquivos
    echo "While backuping files of $source_dir: $counter_erro_i Errors; $counter_warnings_i Warnings; $counter_updated_i Updated; $counter_copied_i Copied ($bytes_copied_i B); $counter_deleted_i Deleted ($bytes_deleted_i B)"
    echo "-------------------------------------------------------"

    for dir in "$source_dir"/{*,.*}; do
        #Resetar contadores internos ao entrar em sub-diretorias
        counter_erro_i=0
        counter_warnings_i=0
        counter_copied_i=0
        counter_deleted_i=0
        counter_updated_i=0
        bytes_deleted_i=0
        bytes_copied_i=0

        if [[ -d $dir ]]; then
            filename="${dir##*/}"
            current_backup_DIR="$backup_dir/$filename"

            if ignore_files "$dir" "${array_ignore[@]}"; then
                continue #Ignorar ficheiros/diretorias com o nome encontrado no ficheiro
            fi

            if [[ -e "$current_backup_DIR" ]]; then  #Verificar existência da sub-diretoria
                if [[ $Check_mode -eq 1 ]]; then
                    echo "backup -c $dir $current_backup_DIR"
                    backup "$dir" "$current_backup_DIR" #Função recursiva à sub-diretoria
                else
                    log $log_file "backup "$dir" "$current_backup_DIR""
                    backup "$dir" "$current_backup_DIR"
                fi
            else
                if [[ $Check_mode -eq 1 ]]; then
                    echo "mkdir -p $current_backup_DIR"
                    mkdir -p "$current_backup_DIR" || { echo "[ERRO] ao criar $current_backup_DIR"; ((counter_erro++)); continue;}   #Criar sub-diretoria
                    echo "Sub-Diretoria $filename criada com sucesso!"
                    backup "$dir" "$current_backup_DIR"
                else
                    mkdir -p "$current_backup_DIR" || { echo "[ERRO] ao criar $current_backup_DIR"; ((counter_erro++)); continue;}
                    echo "Sub-Diretoria $filename criada com sucesso!"
                    log $log_file "mkdir -p "$current_backup_DIR""
                    log $log_file "backup "$dir" "$current_backup_DIR""
                    backup "$dir" "$current_backup_DIR"
                fi
            fi
        fi
    done
```