deti universidade de aveiro
departamento de electrónica, telecomunicações e informática

Projeto 1: Ferramenta de criação/atualização de cópias de segurança em *bash*

Sistemas Operativos

Duarte Gabriel Castro Branco - 119253

2024/2025

Conteúdo

1. Introdução

O objetivo deste projeto foi desenvolver uma ferramenta em ***bash*** para a criação e atualização de cópias de segurança de uma diretoria de trabalho (SOURCE) em outra diretoria de backup (BACKUP). Este processo de backup visa não apenas copiar todos os arquivos e sub-diretórios inicialmente, mas também otimizar as execuções subsequentes, copiando apenas os arquivos que foram modificados ou criados, e removendo do backup aqueles que não existem mais na diretoria de origem.

Além disso, este programa também prevê vários parâmetros opcionais que adicionam flexibilidade à ferramenta, estes são:

- *-c* Modo de teste que apenas mostra o output mas não executa os comandos.
- *-b textfile* Permite a indicação de um ficheiro de texto com uma lista de ficheiros/diretorias que não devem ser copiados.
- *-r regexexp* Apenas serão copiados os ficheiros que verifiquem a *regular expression*.

O desenvolvimento do projeto foi divido em várias etapas progressivas tal como é pedido no guião do mesmo. Neste relatório vou demonstrar e aprofundar a minha implementação de cada uma destas fases num único ficheiro ***backup_tree.sh*** que implementa todas as funcionalidades. Pelo que o meu projeto consiste em apenas 3 ficheiros: ***backup.sh***, ***backup_tree.sh*** e ***backup_tests.sh***.

```
./backup.sh [-c] [-b tfile] [-r regexpr] SOURCE_DIR BACKUP_DIR
```

Código 1.1: *Usage* do projeto

2. backup.sh

2.1 Argumentos de Entrada

Iniciando, vou analisar o *script* ***backup.sh***.

O programa deve começar por filtrar os argumentos de entrada. Esta filtragem é comum ser feita pelo comando integrado ***getopst***, pelo que também usei a mesma metodologia.

No início do *script* criei uma função *usage* que será chamada sempre que o utilizador tenha introduzido algo incorretamente. É uma forma típica de lidar com este tipo de erros.

```
# HELP
usage() {
    echo "Usage: $0 [-c] [-b tfile] [-r regexpr] source_directory
backup_directory"
    exit 1
}
```

Código 2.1: Função *usage*

Criei 3 variáveis que irão armazenar os valores das opções da linha de comandos:

- *check_mode=false* Esta variável será definida como *true* se a opção *-c* for fornecida.
- *exclude_file=' '* Esta variável irá armazenar o caminho para o ficheiro de exclusão, se a opção *-b* for fornecida.
- *regex_pattern=' '* Esta variável irá armazenar o padrão de expressão regular, se a opção *-r* for fornecida.

Depois, uso um ciclo *while* para analisar as opções usando o comando ***getopst***:

- *-c*: Se esta opção for fornecida, a variável *check_mode* é definida como *true*.
- *-b*: Se esta opção for fornecida, o valor que a segue é armazenado na variável *exclude_file*. O *script* também verifica se o ficheiro existe e se contém dados de texto.

- *-r*: Se esta opção for fornecida, o valor que a segue é armazenado na variável *regex_pattern*.
- Se for fornecida uma opção desconhecida, a função *usage* é chamada.

Depois de analisar as opções, o *shift* desloca a lista de argumentos para remover as opções e os seus valores, deixando apenas os argumentos restantes.

Em seguida, verifica-se a existência dos dois argumentos que sobram. Se não existirem, a função *usage* é chamada.

```
check_mode=false
exclude_file=""
regex_pattern=""

# Parse options using getopts
while getopts ":cb:r:" opt; do
    case ${opt} in
        c )
            check_mode=true
            ;;
        b )
            exclude_file=$OPTARG
            if [ ! -f "$exclude_file" ] || ! file "$exclude_file" |
            grep -q "text"; then
                usage
            fi
            ;;
        r )
            regex_pattern=$OPTARG
            ;;
        \? )
            usage
            ;;
    esac
done
shift $((OPTIND -1))

# Checking the remaining arguments
if [ "$#" -ne 2 ]; then
    usage
fi
```

Código 2.2: Filtragem de argumentos

2.2 Diretórios

Por fim, atribuí o primeiro argumento que sobra à variável SOURCE_DIR e o segundo à variável BACKUP_DIR. Estes são os diretórios necessários para o programa funcionar corretamente e fazer todas as operações, os que os torna de extrema importância.

Primeiro é preciso certificar que SOURCE_DIR existe, terminando o programa caso este não seja o caso.

O programa verifica logo a seguir se o BACKUP_DIR existe ou não, criando-o caso não exista. É aqui também que se vê se a opção *-c* foi fornecida pelo utilizador. Caso tenha sido, o programa apenas mostra o output do processo de criação da pasta, mas não a cria.

```
SOURCE_DIR="$1"
BACKUP_DIR="$2"


# Check if source dir exists
if [ ! -d "$SOURCE_DIR" ]; then
    echo "Error: Source directory '$SOURCE_DIR' does not exist."
    exit 1
fi

# Create backup dir if it doesn't exist
if [ ! -d "$BACKUP_DIR" ]; then
    echo -e "Backup directory '$BACKUP_DIR' does not exist. Creating it...\n"
    if [ "$check_mode" = false ]; then
        mkdir -p "$BACKUP_DIR"
    fi
fi
```

Código 2.3: *Check mode*

Incluí também uma opção que usa o programa integrado ***shopt*** para alterar o comportamento da shell e permitir a cópia de ficheiros ocultos. Como não vejo esta funcionalidade muito útil, deixei-a comentada e apenas a escrevi porque um colega mencionou este tipo de situação.

```
# Include hidden files
# shopt -s dotglob
```

Código 2.4: *Hidden files*

Para terminar, importei o programa ***backup_tree.sh*** e chamei a função *backup_tree()* com os argumentos de entrada: SOURCE_DIR, BACKUP_DIR, *check_mode*, *exclude_file* e *regex_pattern*. Vou aprofundar este ficheiro no próximo capítulo.

3. backup_tree.sh

3.1 backup_files.sh e sub-diretorias

Primeiramente, devo referir que o programa ***backup_tree.sh*** é o mais importante deste projeto e foi desenvolvido iterativamente e em diversas fases de acordo com o solicitado no guião. Nesta primeira fase foi pedido para desenvolver código que apenas fazia o backup dos ficheiros e não tinha em consideração sub-diretorias e os respetivos ficheiros. No entanto, após escrever o código - ***backup_files.sh*** - percebi que o problema da deteção de sub-diretorias é facilmente resolvido com uma chamada recursiva da própria função.

Comecei for desenvolver um *loop* que percorra cada *file* existente no SOURCE_DIR. Após identificar o *file* como um ficheiro que existe, o programa copia o *file* para o BACKUP_DIR através do comando integrado ***cp***.

```
# First, process all files in SOURCE_DIR
for file in "$SOURCE_DIR"/*; do
    if [[ -f "$file" ]]; then
        filename=$(basename "$file")
        source_file="$SOURCE_DIR/$filename"
        backup_file="$BACKUP_DIR/$filename"

        # Copy new file to backup
        echo "cp -a '$file' '$backup_file'"

        if [ "$check_mode" = false ]; then
            cp -a "$file" "$backup_file"
        fi
    fi
done
```

Código 3.1: *First loop*

Depois desse *loop* terminar, criei outro *loop* que percorre cada *dir* no SOURCE_DIR. Este serve para identificar os sub-diretórios na SOURCE_DIR. Depois de encontrar um *dir*, a função ***backup_tree.sh*** é chamada outra vez mas com um novo SOURCE_DIR como argumento, sendo este o sub-diretório encontrado.

```
# Now, process all subdirectories in SOURCE_DIR
for dir in "$SOURCE_DIR"/*; do
    if [[ -d "$dir" ]]; then
        filename=$(basename "$dir")
        source_subdir="$SOURCE_DIR/$filename"
        backup_subdir="$BACKUP_DIR/$filename"

        echo -e "$source_subdir is a subdirectory"
        echo -e "Backing up '$source_subdir' to '$backup_subdir'..."

        if [ ! -d "$backup_subdir" ] ||
        [ "$source_subdir" -nt "$backup_subdir" ]; then
            echo "mkdir -p '$backup_subdir'"

            if [ "$check_mode" = false ]; then
                mkdir -p "$backup_subdir"
            fi
        fi

        # Recursively back up the subdirectory
        backup_tree "$source_subdir" "$backup_subdir"
        "$check_mode" "$exclude_file" "$regex_pattern"
    fi
done
```

Código 3.2: *Second loop*

Como já foi referido, o ***backup_tree.sh*** também leva como argumentos as variáveis relativas às opções/argumentos de entrada do utilizador, pelo que estes 2 *loops* têm de os levar em consideração.

Para o argumento *-c* fiz o que fá fiz anteriormente (**ver código 2.3**). Para o argumento *-b* tive de ler o ficheiro que foi dado (caso ele exista) e colocar dentro de um *array* todas as linhas do tal ficheiro, sendo cada linha um elemento do *array*. Para isso usei o comando integrado ***mapfile***. No *loop*, o programa verifica se o *file* está no *array*, passando à frente se assim se verificar.

```
    # Read exclusions into an array if exclude_file is provided
    local exclusions=()
    if [ -n "$exclude_file" ] && [ -f "$exclude_file" ]; then
        mapfile -t exclusions < "$exclude_file"
    fi
    ...
    # Skip if the file is in the exclusions list
    if [[ " ${exclusions[@]} " =~ " $filename " ]]; then
        echo "Skipping '$source_file' as it is in the exclude list."
        continue
    fi
```

Código 3.3: *Handling -b option*

Terminando os argumentos, para o *-r* a implementação também foi simples depois de guardar variável *regex_pattern*, apenas comparei o *file* com o *regex_pattern* e passei à frente se não fossem compatíveis.

```
# Skip if the file does not match the regex pattern
if [[ -n "$regex_pattern" && ! "$filename" =~ $regex_pattern ]]; then
    echo "Skipping '$source_file' as it does not match the regex pattern."
    continue
fi
```

Código 3.4: *Handling -r option*

Terminando este sub-capitulo, é preciso mencionar que este código inicial poderia ser implementado com apenas um *loop*, mas devido ao próximo subcapítulo, decidi que esta era a melhor maneira.

3.2 Detalhes

Embora o código principal já esteja aqui retratado, ainda faltam imensos detalhes que foram pedidos no guião. Por exemplo o que fazer se já existir um ficheiro com o mesmo nome no BACKUP_DIR. No guião é nos pedido para atualizar os ficheiros que foram modificados (os mais recentes). Para isso é preciso alterar o primeiro *loop* e comparar a data de modificação dos ficheiros. Além disso também é preciso implementar o *WARNING* que acontece apenas quando o *backup_file* é mais novo do que o *source_file*.

```
for file in "$SOURCE_DIR"/*; do
    if [[ -f "$file" ]]; then
        # [...] # already existing code
        # Check if backup file exists
        if [ -f "$backup_file" ]; then
            # Check if backup file is newer
            if [ "$backup_file" -nt "$source_file" ]; then
                echo "WARNING: backup entry '$backup_file' is newer
                than '$source_file'; should not happen"
            fi

            # Update if source file is newer
            if [ "$source_file" -nt "$backup_file" ]; then
                echo "'$backup_file' already exists, updating it..."

                if [ "$check_mode" = false ]; then
                    cp -a "$file" "$backup_file"
                fi
            fi
        else
            # Copy new file to backup
            echo "cp -a '$file' '$backup_file'"

            if [ "$check_mode" = false ]; then
                cp -a "$file" "$backup_file"
            fi
        fi
    fi
done
```

Código 3.5: *Same name files in source and backup, and WARNING*

Para além disto, o guião pede para se existir um ficheiro ou diretório no BACKUP_DIR que não exista ou deixou de existir no SOURCE_DIR, deve ser apagado. Ora, para implementar isto é preciso percorrer cada ficheiro do BACKUP_DIR e comparar com o SOURCE_DIR, isto **depois** de ser feito o backup dos ficheiros e **antes** de percorrer o próximo sub-diretório através da recursividade. Por isso é que decidi criar um terceiro *loop*, colocando-o entre o primeiro e o segundo, justificando assim a minha implementação de 2 *loops* ao invés de apenas 1.

O código é parecido com o anterior, embora eu tenha feito uma distinção entre o *file* caso seja um ficheiro ou um diretório. Apenas há uma pequena alteração no processo de ***rm*** (*remove command*), usando o modo recursivo para as pastas, ao contrario dos ficheiros singulares. O modo recursivo funcionaria para os dois casos mas eu prefiro assim.

```
# Check if there are files/dirs in the backup directory that are
# not in the source directory
for file in "$BACKUP_DIR"/*; do
    if [[ -f "$file" ]]; then
        filename=$(basename "$file")
        source_file="$SOURCE_DIR/$filename"
        backup_file="$BACKUP_DIR/$filename"

        # Delete file if it does not exist in the source directory
        if [ ! -f "$source_file" ]; then
            echo "rm '$backup_file'"

            if [ "$check_mode" = false ]; then
                rm "$backup_file"
                deleted=$((deleted + 1))
            fi
        fi
    elif [[ -d "$file" ]]; then
        filename=$(basename "$file")
        source_dir="$SOURCE_DIR/$filename"
        backup_dir="$BACKUP_DIR/$filename"

        # Delete directory if it does not exist in the source directory
        if [ ! -d "$source_dir" ]; then
            echo "rm -r '$backup_dir'"

            if [ "$check_mode" = false ]; then
                rm -r "$backup_dir"
                deleted=$((deleted + 1))
            fi
        fi
    fi
done
```

Código 3.6: *New loop beetween the first and second ones*

3.3 backup_summary.sh

Foi nos pedido para apresentar um sumário no final de ser feito o backup de cada pasta e sub-pasta. Nesse sumário é preciso apresentar o número de erros, número de *warnings*, número de *updates*, tamanho dos ficheiros copiados e tamanho dos ficheiros apagados. Podemos começar por iniciar as variáveis de cada uma destes valores.

```
local errors=0
local warnings=0
local updated=0
local copied=0
local copied_data=0
local deleted=0
local deleted_data=0
```

Código 3.7: *New variables*

Depois decidi criar uma nova função, ***backup_summary***, que apenas mostra o output.

```
backup_summary() {
        echo -e "While backing up $SOURCE_DIR: $errors Errors; $warnings
    Warnings; $updated Updated; $copied Copied ($copied_data B); $deleted
    Deleted ($deleted_data B)\n"
}
```

Código 3.8

Agora, é preciso decidir onde se pode incrementar o valor de cada uma destas variáveis. Após cada comando ***cp*** e ***rm*** é preciso adicionar 1 a cada *copied* e *deleted*, respetivamente. Também é preciso adicionar o tamanho dos ficheiros copiados e apagados, eu fiz isso usando o comando ***du*** (*disk usage*) para mostrar o tamanho em *bytes* e uso o ***cut*** para extrair esse resultado. É de notar que é necessário alterar a variável *deleted_data* antes de executar o comando ***rm***, ao contrário do ***cp*** que funciona das duas formas.

Os *errors* só ocorrem se algo de correr mal ao copiar ou apagar os ficheiros, logo basta fazer um *or statement* quando o ***cp*** e ***rm*** ocorrem e incrementar a variável *errors*.

```
[...]
if [ ! -d "$source_dir" ]; then
    dir_size=$(du -sb "$backup_dir" | cut -f1)
    echo "rm -r '$backup_dir'"

    if [ "$check_mode" = false ]; then
        rm -r "$backup_dir"|| { errors=$((errors + 1)); continue; }
        deleted=$((deleted + 1))
        deleted_data=$((deleted_data + dir_size))
    fi
fi
[...]
if [ "$check_mode" = false ]; then
    cp -a "$file" "$backup_file" || { errors=$((errors + 1)); continue; }
    copied=$((copied + 1))
    copied_data=$((copied_data + $(du -b "$file" | cut -f1)))
fi
```

Código 3.9

Os *Warnings* só devem acontecer quando o ficheiro do BACKUP_DIR é mais recente do que o do SOURCE_DIR. Quando o exato oposto é que se verifica então é preciso fazer o *update* do ficheiro. Isto pode ser implementado, alterando o primeiro *loop*.

Nota: no ficheiro de teste disponibilizado pelo Professor é feita uma distinção entre *update* e *copy*, sendo que um ficheiro que vai ser atualizado não pode também contar como copiado. Tendo isso em conta deixei como comentário a minha implementação anterior.

```
# First, process all files in SOURCE_DIR
for file in "$SOURCE_DIR"/*; do
    if [[ -f "$file" ]]; then
        [...]
        if [ -f "$backup_file" ]; then
            [...]
            # Check if backup file is newer
            if [ "$backup_file" -nt "$source_file" ]; then
                echo "WARNING: backup entry '$backup_file' is newer
                than '$source_file'; should not happen"
                warnings=$((warnings + 1))
            fi
            # Update if source file is newer
            if [ "$source_file" -nt "$backup_file" ]; then
                echo "'$backup_file' already exists, updating it..."
                # NEW ALT due to Professor tests
                # echo "cp -a '$file' '$backup_file'"
                if [ "$check_mode" = false ]; then
                    cp -a "$file" "$backup_file" ||
                    { errors=$((errors + 1)); continue; }
                    # also NEW ALT due to Professor tests
                    # copied=$((copied + 1))
                    # copied_data=$((copied_data + $(du -b "$file" | cut -f1)))
                    updated=$((updated + 1))
                fi
            fi
        else
            [...]
        fi
    fi
done
```

Código 3.10: Incrementing warnings, updates and errors

3.4 backup_check.sh

No guião é pedido para comparar o conteúdo dos ficheiros do BACKUP_DIR com os ficheiros correspondentes do SOURCE_DIR, usando o comando ***md5sum***. Caso se verifique que não são iguais é pedido para ser escrita uma mensagem do tipo: *"src/text.txt bak1/text.txt differ"*.

Esta parte só deve acontecer se o backup_file existir pelo que basta apenas adicionar umas linha de código no primeiro *loop* já existente. O comando ***md5sum*** calcula o *hash* do ficheiro da *source* e (|) tira (***awk*** *command, more advanced than* ***cut***) apenas o primeiro valor do output deste comando uma vez que o output segue este padrão:

"d41d8cd98f00b204e9800998ecf8427e *filename*".

```
# First, process all files in SOURCE_DIR
for file in "$SOURCE_DIR"/*; do
    if [[ -f "$file" ]]; then
        [...]
        # Check if backup file exists
        if [ -f "$backup_file" ]; then
            # backup_check
            # ---
            source_md5=$(md5sum "$source_file" | awk '{ print $1 }')
            backup_md5=$(md5sum "$backup_file" | awk '{ print $1 }')
            if [[ "$source_md5" != "$backup_md5" ]]; then
                echo "$source_file and $backup_file differ."
            fi
            # ---
        [...]
    fi
done
```

Código 3.11: *backup_check*

4. backup_tests.sh

Para provar o bom funcionamento do meu projeto escrevi um pequeno *script* que testa cada um dos possíveis casos de output. É de notar que como pasta de trabalho escolhi o diretório do meu site web pessoal, o que seria um possível caso de uso na vida real. A pasta segue esta estrutura:

```
Personal_web
├── Drawings
│   ├── Jacques_Callot_Study.png
│   ├── Luis_de_Camões_Satues.png
│   ├── Miguel_Angelo_Study.png
│   └── Peter_Paul_Rubens_Study.png
├── aboutme.html
├── art.html
├── cv.html
├── duarte-profile.jpeg
├── global.css
├── index.html
└── project.html
```

Também escrevi um ficheiro de texto - exc - que vai usado com o parâmetro *-b*.

```
art.html
Personal_web/Drawings
```

Agora, no ficheiro de testes comecei por escrever 4 testes dos diretórios, com diferentes parâmetros. Em comentário escrevi o que deveria acontecer para não me confundir. Estes testes funcionaram corretamente, em baixo estão os primeiros testes e a parte do output com mais relevância para os casos.

```
./backup.sh -c "Personal_web" "Personal_backup"
./backup.sh -c -b "exc" "Personal_web" "Personal_backup"
# Should exclude art.html; Personal_web/Drawings
./backup.sh -c -r ".*\.css" "Personal_web" "Personal_backup"
# Should only backup css files
./backup.sh -c -b "exc" -r ".*\.css" "Personal_web" "Personal_backup"
# Should do the two things above
```

```
Personal_web is a directory
Backing up 'Personal_web' to 'Personal_backup'...
cp -a 'Personal_web/aboutme.html' 'Personal_backup/aboutme.html'
cp -a 'Personal_web/art.html' 'Personal_backup/art.html'
cp -a 'Personal_web/cv.html' 'Personal_backup/cv.html'
cp -a 'Personal_web/duarte-profile.jpeg' 'Personal_backup/duarte-profile.jpeg'
cp -a 'Personal_web/global.css' 'Personal_backup/global.css'
cp -a 'Personal_web/index.html' 'Personal_backup/index.html'
cp -a 'Personal_web/project.html' 'Personal_backup/project.html'
While backing up Personal_web: 0 Errors; 0 Warnings; 0 Updated;
0 Copied (0 B); 0 Deleted (0 B)

Personal_web/Drawings is a subdirectory
Backing up 'Personal_web/Drawings' to 'Personal_backup/Drawings'...
mkdir -p 'Personal_backup/Drawings'
cp -a 'Personal_web/Drawings/Jacques_Callot_Study.png' 'Personal_backup/Drawing-
cp -a 'Personal_web/Drawings/Luis_de_Camões_Statue.png' 'Personal_backup/Drawin-
cp -a 'Personal_web/Drawings/Miguel_Angelo_Study.png' 'Personal_backup/Drawings-
cp -a 'Personal_web/Drawings/Peter_Paul_Rubens_Study.png' 'Personal_backup/Draw-
While backing up Personal_web/Drawings: 0 Errors; 0 Warnings; 0 Updated;
0 Copied (0 B); 0 Deleted (0 B)
-------------
Personal_web is a directory
Backing up 'Personal_web' to 'Personal_backup'...
[...]
Skipping 'Personal_web/art.html' as it is in the exclude list.
[...]
Skipping 'Personal_web/Drawings' as it is in the exclude list.
-------------
Personal_web is a directory
Backing up 'Personal_web' to 'Personal_backup'...
Skipping 'Personal_web/aboutme.html' as it does not match the regex pattern.
[...]
cp -a 'Personal_web/global.css' 'Personal_backup/global.css'
Skipping 'Personal_web/index.html' as it does not match the regex pattern.
[...]
While backing up Personal_web: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0 B);
0 Deleted (0 B)
Personal_web/Drawings is a subdirectory
Backing up 'Personal_web/Drawings' to 'Personal_backup/Drawings'...
mkdir -p 'Personal_backup/Drawings'
Skipping 'Personal_web/Drawings/Jacques_Callot_Study.png' as it does not match -
[...]
While backing up Personal_web/Drawings: 0 Errors; 0 Warnings; 0 Updated;
0 Copied (0 B); 0 Deleted (0 B)
```

```
-------------
Personal_web is a directory
Backing up 'Personal_web' to 'Personal_backup'...
Skipping 'Personal_web/aboutme.html' as it does not match the regex pattern.
Skipping 'Personal_web/art.html' as it is in the exclude list.
Skipping 'Personal_web/cv.html' as it does not match the regex pattern.
Skipping 'Personal_web/duarte-profile.jpeg' as it does not match the regex p-
cp -a 'Personal_web/global.css' 'Personal_backup/global.css'
Skipping 'Personal_web/index.html' as it does not match the regex pattern.
Skipping 'Personal_web/project.html' as it does not match the regex pattern.
While backing up Personal_web: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0 B);
0 Deleted (0 B)

Skipping 'Personal_web/Drawings' as it is in the exclude list.
```

Todos os testes correram da forma expectável. A seguir tirei o modo *-c* e corri o programa 2 vezes seguidas, uma após a outra, tendo tido este output:

```
FIRST BACKUP
Personal_web is a directory
Backing up 'Personal_web' to 'Personal_backup'...
cp -a 'Personal_web/aboutme.html' 'Personal_backup/aboutme.html'
cp -a 'Personal_web/art.html' 'Personal_backup/art.html'
cp -a 'Personal_web/cv.html' 'Personal_backup/cv.html'
cp -a 'Personal_web/duarte-profile.jpeg' 'Personal_backup/duarte-profile-
cp -a 'Personal_web/global.css' 'Personal_backup/global.css'
cp -a 'Personal_web/index.html' 'Personal_backup/index.html'
cp -a 'Personal_web/project.html' 'Personal_backup/project.html'
While backing up Personal_web: 0 Errors; 0 Warnings; 0 Updated; 7 Copied
(25763 B); 0 Deleted (0 B)

Personal_web/Drawings is a subdirectory
Backing up 'Personal_web/Drawings' to 'Personal_backup/Drawings'...
mkdir -p 'Personal_backup/Drawings'
cp -a 'Personal_web/Drawings/Jacques_Callot_Study.png' 'Personal_backup/Dr-
cp -a 'Personal_web/Drawings/Luis_de_Camões_Statue.png' 'Personal_backup/D-
cp -a 'Personal_web/Drawings/Miguel_Angelo_Study.png' 'Personal_backup/Dra-
cp -a 'Personal_web/Drawings/Peter_Paul_Rubens_Study.png' 'Personal_backup-
While backing up Personal_web/Drawings: 0 Errors; 0 Warnings; 0 Updated; 4
Copied (5479875 B); 0 Deleted (0 B)

SECOND BACKUP
Personal_web is a directory
Backing up 'Personal_web' to 'Personal_backup'...
While backing up Personal_web: 0 Errors; 0 Warnings; 0 Updated; 0 Copied
(0 B); 0 Deleted (0 B)

Personal_web/Drawings is a subdirectory
Backing up 'Personal_web/Drawings' to 'Personal_backup/Drawings'...
While backing up Personal_web/Drawings: 0 Errors; 0 Warnings; 0 Updated;
0 Copied (0 B); 0 Deleted (0 B)
```

Assim como era esperado, recebemos o sumário com o número e quantidade de ficheiros copiados no primeiro backup, enquanto que no segundo backup nada foi copiado.

Todos estes testes apenas mostram que tudo funciona corretamente quando nada fora do vulgar acontece, mas também temos de testar casos mais inusitados que possam provocar *warnings* e erros, para confirmar que o programa os contabiliza e age de acordo com o guião. Primeiro, criei uma nova pasta e um novo ficheiro na pasta de backup, pasta esta que já tem os ficheiro copiados da *source*. Apaguei também um ficheiro da pasta *source* e corri o programa outra vez.

```
# Delete files/folders that no longer exist in the source directory
touch "Personal_backup/NEWFILE"
mkdir -p "Personal_backup/NEWFOLDER/"
rm "Personal_web/aboutme.html"
./backup.sh "Personal_web" "Personal_backup"
# Should delete NEWFILE and NEWFOLDER and aboutme.html
```

```
Personal_web is a directory
Backing up 'Personal_web' to 'Personal_backup'...
rm 'Personal_backup/aboutme.html'
rm 'Personal_backup/NEWFILE'
rm -r 'Personal_backup/NEWFOLDER'
While backing up Personal_web: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0 B);
3 Deleted (2934 B)

Personal_web/Drawings is a subdirectory
Backing up 'Personal_web/Drawings' to 'Personal_backup/Drawings'...
While backing up Personal_web/Drawings: 0 Errors; 0 Warnings; 0 Updated;
0 Copied (0 B); 0 Deleted (0 B)
```

E tal como esperado, tanto a pasta e o ficheiro criados no backup, tanto o ficheiro que foi apagado da *source*, foram apagados e devidamente contabilizados.

Vou testar agora se a implementação do ***backup_check*** funciona corretamente. Testei dois casos, primeiro altero um ficheiro já existente no backup, o que deve provocar um *update*, depois altero um ficheiro no *source* que já existia no backup, o que deve provocar um *warning*. Ambos os casos devem referir que os ficheiros diferem.

```
# backup check
echo "TEST" | cat > "Personal_web/art.html"
./backup.sh "Personal_web" "Personal_backup"
# Should update art.html
echo "TEST" | cat > "Personal_backup/cv.html"
./backup.sh "Personal_web" "Personal_backup"
# Should get a Warning
```

```
Personal_web is a directory
Backing up 'Personal_web' to 'Personal_backup'...
Personal_web/art.html and Personal_backup/art.html differ.
'Personal_backup/art.html' already exists, updating it...
While backing up Personal_web: 0 Errors; 0 Warnings; 1 Updated;
1 Copied (5 B); 0 Deleted (0 B)

Personal_web/Drawings is a subdirectory
Backing up 'Personal_web/Drawings' to 'Personal_backup/Drawings'...
While backing up Personal_web/Drawings: 0 Errors; 0 Warnings; 0 Updated;
0 Copied (0 B); 0 Deleted (0 B)

Personal_web is a directory
Backing up 'Personal_web' to 'Personal_backup'...
Personal_web/cv.html and Personal_backup/cv.html differ.
WARNING: backup entry 'Personal_backup/cv.html' is newer than
'Personal_web/cv.html'; should not happen
While backing up Personal_web: 0 Errors; 1 Warnings; 0 Updated;
0 Copied (0 B); 0 Deleted (0 B)

Personal_web/Drawings is a subdirectory
Backing up 'Personal_web/Drawings' to 'Personal_backup/Drawings'...
While backing up Personal_web/Drawings: 0 Errors; 0 Warnings; 0 Updated;
0 Copied (0 B); 0 Deleted (0 B)
```

Outra vez o output é o esperado. ☺

4.1 Testes do professor

No ficheiro de testes do professor é feito um backup de uma pasta *src* numa *backup_test* com ficheiros específicos. No final do programa usa-se o comando ***ls*** para verificar se as 2 pastas mantêm a mesma estrutura, incluindo as sub pastas. Isso foi assegurado pelo meu programa e correu como devia, garantido logo a *score* de 40%. No entanto, a primeira parte não foi contabilizada, o que achei estranho. Decidi, então, eliminar a frase do programa de testes que apaga o output do meu programa e comparei o meu output com o output pretendido.

```
src is a directory
Backing up 'src' to 'backup_test'...
'backup_test/output.txt' already exists, updating it...
cp -a 'src/start.sh' 'backup_test/start.sh'
While backing up src: 0 Errors; 0 Warnings; 1 Updated; 1 Copied (8304 B);
0 Deleted (0 B)

src/aaa is a subdirectory
Backing up 'src/aaa' to 'backup_test/aaa'...
While backing up src/aaa: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0 B);
0 Deleted (0 B)
```

Código 4.1: Meu output

```
cp -a src/output.txt backup_test/output.txt
cp -a src/start.sh backup_test/start.sh
While backuping src: 0 Errors; 0 Warnings; 1 Updated; 1 Copied (8304B);
0 Deleted (0B)
While backuping src/aaa: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B);
0 Deleted (0B)
```

Código 4.2: Output esperado

Como dá para perceber, é apenas uma questão de formatação, uma vez que ambos os programas escrevem os mesmos comandos e entregam o mesmo sumário. ☺

5. Bibliografia

Para realizar este projeto foi preciso recorrer à documentação/manuais de vários programas integrados da bash como: getopst; shopt; cp; mapfile; rm, awk.

Devo também mencionar que este programa é muito semelhante a outro programa que já uso há algum tempo: **rsync** (*remote sync*). É uma utilidade escrita em C em 1996 e é usada para sincronizar servidores remotos, sendo mantida até hoje. Sempre gostei muito deste programa, pelo que me inspirei nele para fazer este projeto.