universidade de aveiro deti departamento de electrónica, telecomunicações e informática

FERRAMENTA DE CRIAÇÃO/ATUALIZAÇÃO DE CÓPIAS DE SEGURANÇA EM BASH

Trabalho realizado por:

Gonçalo Almeida, 119792

Igor Baltarejo, 118832

Sistemas Operativos

Prof. Nuno Lau

Ano Letivo 2024/2025

Índice

1. Introdução

Ao longo deste relatório será descrito em que consiste o projeto, a abordagem tomada para a resolução do problema dado, as várias etapas de desenvolvimento da nossa solução, possíveis problemas e as devidas soluções que surgiram, explicado o código e apresentado os outputs dos *scripts* desenvolvidos.

Sendo assim, o objetivo do projeto é o desenvolvimento de um programa que, ao ser executado, faça a cópia de todos os ficheiros e subdiretorias que façam parte de uma diretoria de trabalho para uma diretoria de backup. É também necessário que, após a primeira execução, que as próximas façam apenas a cópia de novos ficheiros que tenham surgido na diretoria de trabalho ou de ficheiros que tenham sido atualizados, para a diretoria de *backup*, de forma a aumentar a eficiência e rapidez do programa. O programa deve ainda aceitar as *flags*: **-c**, **-b** e **-r**, que oferecem funcionalidades adicionais, explicadas em detalhe mais à frente.

Uso: backup.sh [-c] [-b tfile] [-r regexpr] dir_trabalho dir_backup.

Foi proposto que o desenvolvimento consistisse em 4 *scripts* em *BASH* dos quais os primeiros três seriam versões melhoradas e construídas em cima da anterior, e o último um *script* que embora relacionado com o projeto, não tivesse por base nenhum dos outros *scripts*, respetivamente: **backup_files.sh**, **backup.sh**, **backup_summary.sh** e **backup_check.sh**.

2. Desenvolvimento

Primeiramente serão mencionadas funções e código comum aos 4 *script*s. Após isso será descrito de que forma abordamos a resolução de cada um dos *scripts* mencionados na introdução, com porções de código para auxiliar a explicação. Será também mencionado outras formas para resolver certos problemas, e porque é que, após deliberar sobre o assunto, escolhemos o caminho que escolhemos.

2.1 Features comuns

Environment

Dois dos requerimentos para todos os *scripts* são: o funcionamento em várias máquinas *Linux* sem que houvesse problemas de compatibilidade graças à existência de diferentes *locales*; e a inclusão de ficheiros escondidos (também conhecidos como ficheiros com ponto, do inglês *dotfiles*). Para garantir isto, usamos os comandos builtin da *BASH* **shopt** e **export.**

```
1  #!/bin/bash
2
3  #Ensures locale is set to C standard
4  export LC_ALL=C
5
6  #Globing now includes "dot files"
7  shopt -s dotglob
```

Figura 1 - Opções da shell e de ambiente (Comum a todos os scripts)

Na linha 2 usamos o comando **export** para definir uma variável de ambiente, que diz á *BASH* para usar o *locale* padrão da linguagem C, escolhemos este pois é um *locale* pré instalado em todas as distribuições de *Linux*, garantindo assim que não há problemas de compatibilidade em termos de datas de modificação e caracteres presentes nas *strings*. Já o comando **shopt** (*shell option*), com a opção -s define que os ficheiros com ponto devem ser incluídos no momento da expansão de expressões de *globbing*.

Variáveis

Definimos variáveis globais para guardar os estados das opções e também o caminho completo da diretoria de trabalho e *backup* originais.

```
#Vars
_checking=false
_help=false
_regex=false
_file=false
_workdir=""
_backupdir=""
```

Figura 2 - Definição de variáveis globais (Comum a todos os scripts)

Flags

Segue-se uma pequena explicação para cada funcionalidade esperada quando usada a respetiva *flag* na chamada das *scripts*:

-h : não é feito o *backup*, imprime uma mensagem de ajuda. (Não requerida pelo guião)
-c : não é feito o *backup*. imprime todos os comandos relacionados com a cópia de ficheiros e directorias que seriam executados pelo *script*.
-b tfile : ignora todos os ficheiros e diretorias incluídas no ficheiro **tfile**
-r regexpr : apenas copia ficheiros cujo nome respeite o padrão de *regex* **regexpr.**

Conjunção das flags -b e -r

Ao longo do projeto percebeu-se que a consideração do comportamento da conjunção destas *flags* era importante para definir uma solução cujo funcionamento fosse consistente e previsível. Chegamos a conclusão que faz mais sentido a *flag* -b se sobrepor e fazer com que o programa ignore um ficheiro mesmo que este tenha um nome que respeite a expressão de *regex* da *flag* -r. Para atingir este efeito verificamos sempre primeiro se o ficheiro/diretoria deve ser ignorado e só depois com o *regex* no caso dos ficheiros.

Flag/argument parsing

Embora ao longo das iterações do projeto o código para a análise de argumentos foi se tornando mais completo, o código base para tal usa a função **getopts** para verificar se determinadas *flags* estavam presentes na chamada ao *script*. Caso assim seja, as variáveis previamente declaradas para cada uma das flags tomarão o valor de ***"true"***. Este valor é importante e será explicado mais à frente.

```
39  #Argument and flag parsing
40  while getopts ":ch" flag
41  do
42      case $flag in
43          c)
44              _checking=true ;;
45          h)
46              _help=true ;;
47          ?)
48              echo "Invalid option -$flag: aborting backup"
49              exit 1 ;;
50      esac
51  done
52
53  #Strip flags and flag arguments from argument list
54  shift $(($OPTIND - 1))
```

Figura 3 - Solução para a análise de flags e argumentos (backup_files.sh)

Na linha 41, a chamada ***optstring*** começa com ":", isto sinaliza ao comando **getopts** que deve ser executado no modo silencioso, o que faz com que o comando não imprima nenhuma mensagem de erro quando este encontra uma flag não suportada (que não se encontre na ***optstring***), permitindo nos assim resolver este caso como quisermos. O **getopts** guarda na variável **flag** o valor da *flag*, "c" para a *flag* **-c**, "h" para a *flag* -h, por aí em diante para todas as flags na chamada até não existir mais nenhuma (As não suportadas equivalem a "?").

Contudo este processo não é perfeito, uma vez que assumimos que todas as *flags* se encontram antes dos parâmetros posicionais. Ao contrário do que acontece com a maioria dos comandos da *BASH* e de outros pacotes de utilitários, em que as *flags* podem se encontrar no início, fim ou até mesmo entre parâmetros posicionais.

As variáveis **OPTIND** e **OPTARG** são variáveis de ambiente usadas pelo **getopts** para a indicação da posição do próximo parâmetro a ser processado e o argumento passado à *flag* respetivamente.

```
146 #Argument and flag parsing
147 while getopts ":chb:r:" flag
148 do
149     case $flag in
150         c)
151             _checking=true ;;
152         h)
153             _help=true ;;
154         b)
155             _tfile="$OPTARG"
156             _file=true ;;
157         r)
158             _regexpr="$OPTARG"
159             _regex=true ;;
160         ?)
161             echo "Invalid option -$flag: aborting backup"
162             exit 1 ;;
163     esac
164 done
165
166 #Strip flags and flag arguments from argument list
167 shift $(($OPTIND - 1))
```

Figura 4 - Exemplo da evolução da solução (backup_summary.sh)

Para além disso verificamos a validade dos vários argumentos passados às *flags,* e dos parâmetros posicionais. A versão mostrada em diante é referente ao *script* **backup_summary.sh**, o mais completo. As únicas diferenças comparativamente aos *scripts* **backup_files.sh** e **backup_check.sh** é a verificação dos argumentos das *flags* **-b** e **-r**, **_file** e **_regexpr** respetivamente

Estas verificações são básicas, garantem que: ficheiros e diretorias são válidos(as); são passados ambos os argumentos posicionais; e o padrão de *regex* é válido e que a diretoria de trabalho não contém a diretoria de *backup*. Quero salientar a utilização das variáveis associadas às *flags*, que apesar de não conterem um valor booleano, contém os valores “false” ou “true”, comandos da *BASH* que retornam 1 e 0 respetivamente quando utilizados em conjunção com o *if* e outras expressões na *BASH*.

```
169 if $_help
170 then
171     echo "Usage: $1 [-c] [-b tfile] [-r regexpr] workingDir backupDir"
172     exit 0
173 fi
174
175 if $_file
176 then
177     if [[ ! -f $_tfile ]]
178     then
179         echo "Bad argument for -b: '$_tfile' is not a file or doesn't exist"
180         exit 1
181     fi
182 fi
183
184 if $_regex
185 then
186     #Using grep as a pattern validator
187     #If grep has a bad regular expression exit code should be 2
188     echo "2005" | grep -P "$_regexpr" &>/dev/null
189
190     if [[ $? -eq 2 ]]
191     then
192         echo "Bad argument for -r: '$_regexpr' isn't a valid regex expression"
193         exit 1
194     fi
195 fi
196
197 #Ensure no missing or excess argument
198 if [[ $# -ne 2 ]]
199 then
200     case "$#" in
201         0) echo "Missing workdir argument" ;;
202         1) echo "Missing backupdir argument" ;;
203         *) echo "Too many arguments" ;;
204     esac
205     exit 1
206 fi
207
208 #Check for the existence of the workDir
209 if [[ ! -d "$1" ]]
210 then
211     echo "Bad argument: '$1' is not a directory or doesn't exist"
212     exit 1
213 fi
214
215 #Resolve full paths
216 _workdir=$(realpath "$1")
217 _backupdir=$(realpath "$2")
218
219 #Check if backupDir is a subdirectory of workingDir
220 if [[  "${_backupdir##$_workdir}" != "$_backupdir" ]]
221 then
222     echo "Error: Backup directory is a sub-directory of working directory"
223     exit 1
224 fi
```

Figura 5 - Solução para a validação de argumentos (backup_summary.sh)

Funções de suporte

Com o objetivo de melhor a legibilidade e tornar mais fácil a adaptação do código para quando a *flag* **-c** fosse utilizada, criamos as seguintes funções:

```
17  #Helper functions for use of _checking
18  function cpHelper() {
19      echo "cp -a $(basename $_workdir)${1##$_workdir} $(basename $_backupdir)${2##
    $_backupdir}"
20      $_checking || cp -a "$1" "$2" &>/dev/null
21
22      if [[ $? -ne 0 ]]
23      then
24          echo "ERROR: couldn't copy $(basename $_workdir)${1##$_workdir}"
25      fi
26
27      return 0
28  }
29
30  function mkdirHelper() {
31      echo "mkdir $(basename $_backupdir)${1##$_backupdir}"
32      $_checking || mkdir "$1" &>/dev/null
33
34      if [[ $? -ne 0 ]]
35      then
36          echo "ERROR: couldn't create directory $(basename $_workdir)${1##$_workdi
    r}"
37      fi
38
39      return 0
40  }
41
42  function rmHelper() {
43      $_checking || rm -r "$1" &>/dev/null
44
45      return 0
46  }
```

Figura 6 - Funções de suporte (backup.sh)

Qualquer uma destas funções permite a separação da lógica dos comandos de cópia e remoção de ficheiros, da lógica envolvente na travessia do sistema de ficheiros. Nelas usamos "&>/dev/null" para redirecionar o stderr e o stdout para o dispositivo /dev/null, suprimindo as mensagens de erro e eventuais outputs dos comandos. Nas condições que nos permitem executar os comandos que interagem com o sistema de ficheiros, tiramos proveito da propriedade da avaliação em curto circuito das expressões booleanas em *BASH*. Esta propriedade será usada múltiplas vezes ao longo do projeto para evitar o uso excessivo de expressões *if-then*, reduzindo o tamanho e aumentando a legibilidade do código. Mais à frente no relatório será mostrado como estas funções evoluíram.

2.2 backup_files.sh

Esta primeira versão do projeto tem a capacidade de fazer *backup* de ficheiros, da diretoria de trabalho para a de *backup* e aceita uma *flag*, **-c**, que quando usada, indica ao *script* que deve apenas listar os comandos que iria executar caso fizesse o *backup*.

Para isso percorremos pelos ficheiros/diretorias resultantes da expansão da expressão "$_workdir"/*, ignorando as diretorias e fazendo apenas a cópia de ficheiros novos ou atualizados. Consideramos excepcional a situação em que o ficheiro existe em ambas as diretorias sendo mais recente na de *backup*, não alterando o ficheiro e alertando o utilizador. A cópia dos ficheiros constitui metade da solução, a outra metade passa por percorrer a diretoria de *backup*, já criada anteriormente, e remover ficheiros/diretorias que não pertencem à diretoria de trabalho.

```
85  #Copy/Update
86  for fpath in "$_workdir"/*
87  do
88      fname=$(basename "$fpath")
89
90      #Check if directory is not empty
91      [[ "$fname" == "*" ]] && break
92
93      if [[ -d "$fpath" ]]
94      then
95          continue
96      fi
97
98      if [[ ! -f "$_backupdir/$fname" ]] || [[ "$fpath" -nt "$_backupdir/$fname" ]]
99      then
100         cpHelper "$fpath" "$_backupdir/$fname"
101     elif [[ "$fpath" -ot "$_backupdir/$fname" ]]
102     then
103         echo "WARNING: backup entry $(basename $_workdir)${fpath##$_workdir} is n
    ewer than $(basename $_backupdir)${2##$_backupdir}; Should not happen"
104     fi
105 done
106
107 #Remove files not in workdir
108 for fpath in "$_backupdir"/*
109 do
110     fname=$(basename "$fpath")
111
112     #Check if directory is not empty
113     [[ "$fname" == "*" ]] && break
114
115     if [[ ! -e "$_workdir/$fname" ]]
116     then
117         rmHelper "$fpath"
118     fi
119 done
```

Figura 7 - Solução para percorrer os ficheiros/diretorias (backup_files.sh)

2.3 backup.sh

Esta versão seguinte, **backup.sh**, foi construída a partir da anterior, adicionando em cima as funcionalidades associadas às *flags* **-b** e **-r**, e capacidade de copiar ficheiros e diretorias recursivamente. Serão, portanto, analisadas as melhorias feitas à iteração anterior:

Iteração sob diretórios recorrendo à recursão

A escolha da recursão, ainda que sugerida no guião, era a mais óbvia devido à natureza arbórea dos sistemas de ficheiros. Para a implementar foi tão simples como usar a solução da iteração anterior e invés de ignorar as diretorias, chamamos a função de *backUp* nas mesmas. A função de *backUp* contém a solução utilizada no *script* **backup_files.sh** com esta pequena alteração.

```
81  if [[ -d "$fpath" ]]
82  then
83      backUp "$fpath" "$backupdir/$fname"
84  elif [[ ! -f "$backupdir/$fname" ]] || [[ "$fpath" -nt "$backupdir/$fname" ]]
85  then
86
87      $_regex && [[ ! "$fname" =~ $_regexpr ]] && continue
88
89      cpHelper "$fpath" "$backupdir/$fname"
90
91  elif [[ "$fpath" -ot "$backupdir/$fname" ]]
92  then
93      echo "WARNING: backup entry $(basename $_workdir)${fpath##$_workdir} is newer
    than $(basename $_backupdir)${2##$_backupdir}; Should not happen"
94  fi
```

Figura 8 - Condição para a recursão (backup.sh)

A função *backUp* cria (caso ainda não exista) uma diretoria de *backup* com o mesmo nome do diretório que está a ser copiado, mas dentro do diretório de destino de *backup*. Dessa forma, preservamos a estrutura de diretórios original.

Flags -b e -r

A implementação de ambas funcionalidades passa por adicionar duas condições. A primeira associada a *flag* **-r,** pode ser vista na linha 87 da figura 8, se a *flag* estava presente e a expressão de *regex* não é respeitada ignoramos aquele ficheiro. A segunda, associada a *flag* **-b** pode ser vista na figura 9, se a *flag* **-b** estava presente e o ficheiro/diretoria deve ser ignorado, ignoramos-o(a).

```
75  if $_file && bFiltering "$fpath"
76  then
77      echo "$(basename $_workdir)${fpath##$_workdir} ignored"
78      continue
79  fi
```

Figura 8 - Condição para a recursão (backup.sh)

```
48  function bFiltering() {
49      local fpath=$1
50      local relPath=${fpath##$_workdir/} #passar fpath
51      local grepstr="$(grep -i -E "^($_workdir)?/?$relPath$" "$_tfile")"
52      if [[ "$grepstr" == "$relPath" ]] || [[ "$grepstr" == "$fpath" ]]
53      then
54          return 0
55      fi
56      return 1
57  }
```

Figura 9 - Função bFiltering (backup.sh)

A função apresentada aceita um caminho absoluto para um ficheiro, transformando-o num caminho relativo à diretoria de trabalho. Em seguida, utiliza o comando grep para verificar se o caminho absoluto, ou relativo do ficheiro está presente no *tfile*. Apesar de parecer redundante à primeira vista, esse processo permite que tanto caminhos relativos quanto absolutos sejam pesquisados no *tfile*, aumentando a robustez da solução. Por fim, a função retorna 0 se o ficheiro estiver em tfile, ou 1 caso contrário.

2.4 backup_summary.sh

Esta é a última versão do que podemos dizer que foi o processo de construção de uma ferramenta de criação/atualização de ficheiros de *backup*. Este código, comparado ao anterior, terá um atributo diferenciador: A capacidade de, para cada diretoria que seja copiada, escrever na consola o número de erros, *warnings*, ficheiros atualizados, apagados, copiados e tamanho de ficheiros copiados e apagados.

Estrutura de dados

Para guardar o número de erros, warnings, etc… A nossa solução foi criar uma lista associativa, uma estrutura de dados nativa do *BASH,* que têm um comportamento similar a um *map*, dicionário ou *hashtable* noutras linguagens de programação.

```
94  #Local summary dic, can be accessed by functions called here
95      local -A summary=(['errors']=0 ['warnings']=0 ['num_updated']=0 ['num_copied']
    =0 ['num_removed']=0 ['size_removed']=0 ['size_copied']=0)
```

Figura 10 - Lista associativa em BASH (backup_summary.sh)

Para mostrar na consola os dados para cada diretoria usamos a seguinte função:

```
24  function printSummary() {
25      echo "While backuping $(basename $_workdir)${workdir##$_workdir}: ${summary[er
    rors]} Errors; ${summary[warnings]} Warnings; ${summary[num_updated]} Updated; ${s
    ummary[num_copied]} Copied (${summary[size_copied]}B); ${summary[num_removed]} Del
    eted (${summary[size_removed]}B)"
26  }
```

Figura 11 - Função que acede lista associativa e imprime os valores na consola (backup_summary.sh)

Foi também criada uma função (*summaryAdd*) para adicionar mais rapidamente valores inteiros em cada *key* da lista associativa.

```
17  # usage: summaryAdd key amount
18  function summaryAdd() {
19      # adds amount to key in summary dic
20      local key=$1; local amount=$2
21      summary[$key]=$(( summary[$key] + $amount ))
22  }
```

Figura 12 - Função summaryAdd (backup_summary.sh)

Aqui está uma tabela com os acontecimentos que são mencionados no sumário e as situações que os geram:

Erros	Se a remoção/cópia de ficheiro não for bem sucedida, ou a criação de uma diretoria. Possivelmente devido a problemas com permissões ou espaço livre.
Warnings	Quando um ficheiro da diretoria de *backup* tem uma data de modificação mais recente que a sua *counterpart* na diretoria de trabalho.
Atualizados	Quando ficheiros que já existem na diretoria de *backup* são atualizados, substituídos por uma versão mais recente da diretoria de trabalho
Copiados	Quando ficheiros são copiados da diretoria de trabalho para a de *backup.*
Removidos	Quando os ficheiros são removidos da diretoria de *backup*, quando a sua versão original já não se encontra na diretoria de trabalho.
Tamanho copiado	Soma do tamanho, em bytes, dos ficheiros copiados.
Tamanho removido	Soma do tamanho, em bytes, dos ficheiros removidos.

Tabela 1 - Explicação da contagem de dados

Evolução das funções de suporte

Como falamos anteriormente estas funções foram bastante úteis para a clareza do código e a implementação das funcionalidade adicionais requeridas nesta iteração do projeto foram bastante simplificadas, requereu apenas a correta adição de inteiros a cada key da lista associativa dependendo do resultado das funções de remover e copiar ficheiros e criar diretorias, e para recolher a informação sobre o tamanho de cada ficheiro usamos o comando **stat** da *BASH*.

```
28  function cpHelper() {
29      echo "cp -a $(basename $_workdir)${1##$_workdir} $(basename $_backupdir)${2##
    $_backupdir}"
30
31      local size=$(stat -c %s "$1")
32      local updated=false
33      [[ "$1" -nt  "$2" && -f "$2" ]] && updated=true
34
35      $_checking || cp -a "$1" "$2" &>/dev/null
36
37      if [[ $? -ne 0 ]]
38      then
39          echo "ERROR: couldn't copy $(basename $_workdir)${1##$_workdir}"
40          summaryAdd errors 1
41      else
42          $updated && summaryAdd num_updated 1 || {
43              summaryAdd num_copied 1;
44              summaryAdd size_copied $size;
45          }
46      fi
47
48      return 0
49  }
```

Figura 14.1 - Exemplo da evolução das funções de suporte (backup_summary.sh)

```
51  function mkdirHelper() {
52      echo "mkdir $(basename $_backupdir)${1##$_backupdir}"
53      $_checking || mkdir "$1" &>/dev/null
54
55      if [[ $? -ne 0 ]]
56      then
57          echo "ERROR: couldn't create directory $(basename $_backupdir)${1##$_backu
    pdir}"
58          summaryAdd errors 1
59      fi
60
61      return 0
62  }
```

Figura 14.2 - Exemplo da evolução das funções de suporte (backup_summary.sh)

```
64  function rmHelper() {
65      local size=$(stat -c %s "$1")
66      $_checking || rm -r "$1" &>/dev/null
67      if [[ $? -ne 0 ]]
68      then
69          summaryAdd errors 1
70      else
71          summaryAdd num_removed 1
72          summaryAdd size_removed $size
73      fi
74
75      return 0
76  }
```

Figura 14.3 - Exemplo da evolução das funções de suporte (backup_summary.sh)

2.5 backup_check.sh

Este último *script* proposto no guião do projeto tem como objetivo, usando o comando md5sum do *BASH*, verificar se os ficheiros do diretório de trabalho e de *backup* são iguais em termos de conteúdo.

Este programa preserva a lógica de iterar pelo diretório recursivamente, do backup.sh, já no *parsing*/validação das *flags* houve uma redução significativa das verificações feitas.

```
14  function compareFiles() {
15      #Returns 0 if contents are equal, 1 if not
16      #Using md5sum
17      #Output of md5sum: "checksum fname"
18      #Transform output in to an array and compare first element (checksum)
19
20      sum1=($(md5sum "$1")); sum2=($(md5sum "$2"))
21      return [[ "${sum1[0]}" == "${sum2[0]}" ]]
22  }
```

Figura 15 - Função compareFiles (backup_summary.sh)

Esta função transforma o output num *array*, em que o primeiro elemento é a *hash* produzida pelo comando e o segundo é o nome do ficheiro, para o efeito só interessa comparar as *hash* strings e caso sejam iguais o conteúdo dos ficheiros é igual.

```
28  function backupCheck() {
29      local workdir="$1"
30      local backupdir="$2"
31
32      for fpath in "$workdir"/*
33      do
34          fname=$(basename "$fpath")
35
36          [[ "$fname" == "*" ]] && break
37
38          [[ -d "$fpath" ]] && backupCheck "$fpath" "$backupdir/$fname" && continue
39
40          if [[ ! -f "$backupdir/$fname" ]]
41          then
42              echo "${backupdir##$_backupdir}/$fname doesn't exist"
43              continue
44          fi
45
46          compareFiles "$fpath" "$backupdir/$fname"
47          if [[ ! $? -eq 0 ]]
48          then
49              echo "$(basename $_workdir)${fpath##$_workdir} $(basename $_backupdir)
    ${backupdir##$_backupdir}/$fname differ"
50          fi
51      done
52  }
```

Figura 16 - Função backupCheck (backup_summary.sh)

Como podemos ver a lógica dedicada a percorrer o sistema de ficheiros mantém-se idêntica, invès de copiar para cada ficheiro na diretoria de cada iteração, o *compareFiles* faz a sua comparação e dependendo do valor de retorno da linha 46, figura 16, é ou não impresso no terminal uma mensagem para o utilizador.

3.Testes

Para testar a nossa solução, fomos fazendo vários testes ao longo das várias iterações do projeto. Porém neste relatório, por uma questão de brevidade e uma vez que cada iteração foi construída a partir das outras, iremos nos focar nos testes usando a *script* **backup_summary.sh** uma vez que esta é a mais completa, e abrange todas as funcionalidades desenvolvidas. Os testes serão demonstrados através de uma print da consola, com o comando e subsequente resultado e estrutura de ambas as diretorias na seguinte ordem, trabalho e *backup*, salvo algum caso especial, no qual será esclarecido a diferença.

Para a nossa diretoria de trabalho escolhemos uma mistura complexa de ficheiros e diretorias vazios(as), com espaços no nome e escondidos(as) (com o ponto a anteceder o nome). Isto com o objetivo de tentar abranger o máximo de *edge cases* e procurar ter um código robusto

3.1 *Backup*

Backup de ficheiros e diretórios

Neste teste será também apresentado o esquema de ficheiros e subdiretorias da diretoria a que pertence o diretório de trabalho e de *backup*, para verificar que a diretoria de *backup* foi criada, ao ter sido detetado que ainda não existia.

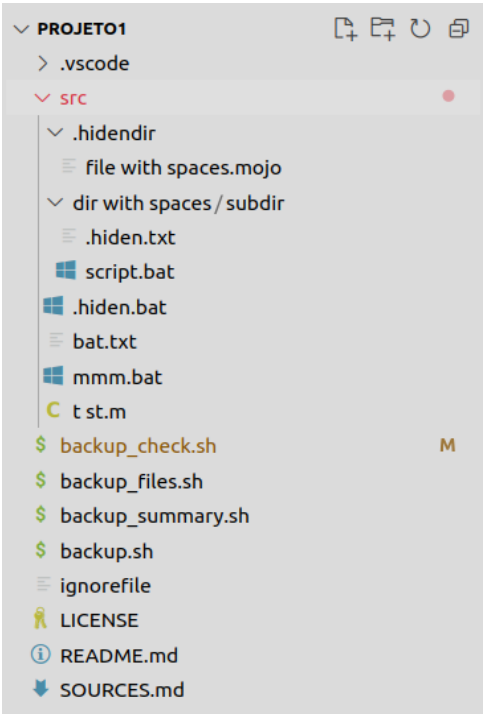

Figura 17.1 - Estrutura de "PROJETO1" pré execução de programa

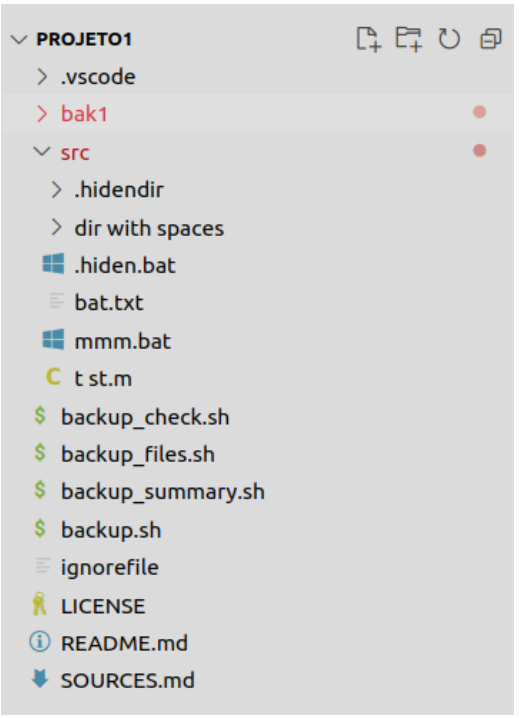

Figura 17.2 - Estrutura de "PROJETO1" pós execução de programa

Nota-se então, ao observar as figuras 17.1 e 17.2, que a criação da diretoria de *backup* foi bem sucedida, pois não existia antes da execução do *script*.

```
 % ./backup_summary.sh src bak1
mkdir bak1
cp -a src/.hiden.bat bak1/.hiden.bat
mkdir bak1/.hidendir
cp -a src/.hidendir/file with spaces.mojo bak1/.hidendir/file with spaces.mojo
While backuping src/.hidendir: 0 Errors; 0 Warnings; 0 Updated; 1 Copied (1432B); 0 Deleted (0B)
cp -a src/bat.txt bak1/bat.txt
mkdir bak1/dir with spaces
mkdir bak1/dir with spaces/subdir
cp -a src/dir with spaces/subdir/.hiden.txt bak1/dir with spaces/subdir/.hiden.txt
cp -a src/dir with spaces/subdir/script.bat bak1/dir with spaces/subdir/script.bat
While backuping src/dir with spaces/subdir: 0 Errors; 0 Warnings; 0 Updated; 2 Copied (2866B); 0 Deleted (0B)
While backuping src/dir with spaces: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
cp -a src/mmm.bat bak1/mmm.bat
cp -a src/t st.m bak1/t st.m
While backuping src: 0 Errors; 0 Warnings; 0 Updated; 4 Copied (5732B); 0 Deleted (0B)
```

Figura 17.3 - Output do comando ./backup_summary src bak1

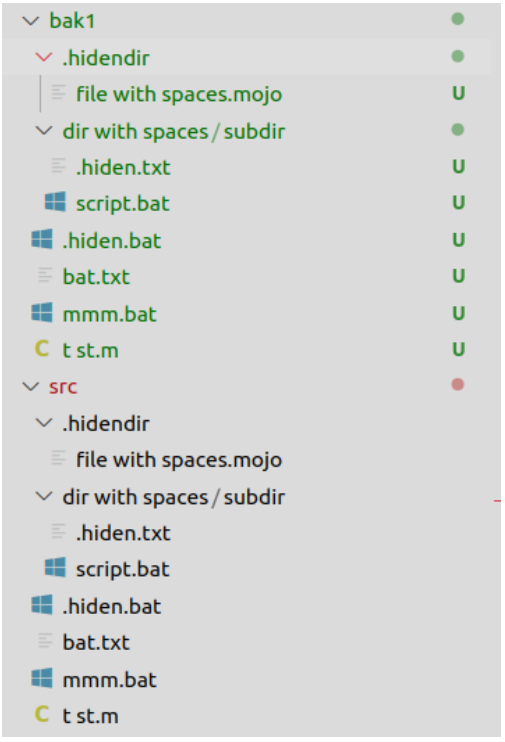

Figura 17.4 - Estrutura src e bak1

Nota-se então, ao observar a figura 17.4, que o *backup* foi bem sucedido, visto que ambos diretórios, *src* e *bak1,* estão iguais.

3.2 *Flags*

Uso da flag -c

Apagando o diretório backupdir e voltando a executar o código, agora com a *flag* **-c** temos os seguintes resultados:

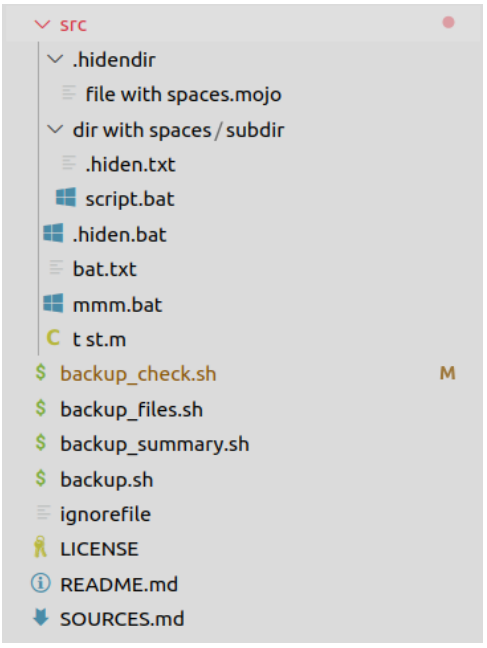

Figura 18.1 - Estrutura de "PROJETO1"
pré execução de programa

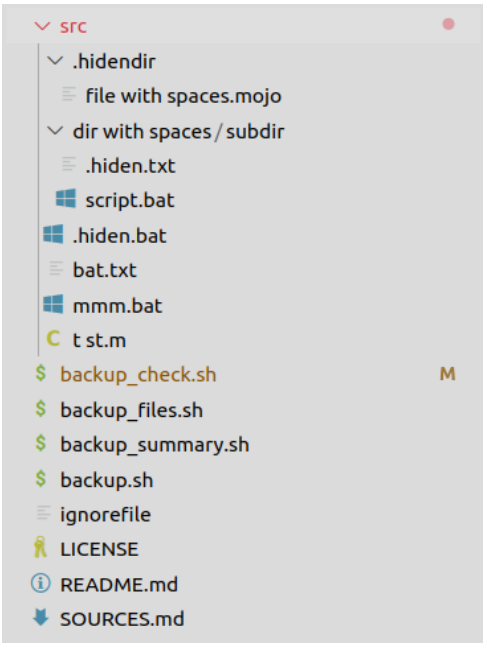

Figura 18.2 - Estrutura de "PROJETO1"
pós execução de programa

```
 % ./backup_summary.sh -c src bak1
mkdir bak1
cp -a src/.hiden.bat bak1/.hiden.bat
mkdir bak1/.hidendir
cp -a src/.hidendir/file with spaces.mojo bak1/.hidendir/file with spaces.mojo
While backuping src/.hidendir: 0 Errors; 0 Warnings; 0 Updated; 1 Copied (1432B); 0 Deleted (0B)
cp -a src/bat.txt bak1/bat.txt
mkdir bak1/dir with spaces
mkdir bak1/dir with spaces/subdir
cp -a src/dir with spaces/subdir/.hiden.txt bak1/dir with spaces/subdir/.hiden.txt
cp -a src/dir with spaces/subdir/script.bat bak1/dir with spaces/subdir/script.bat
While backuping src/dir with spaces/subdir: 0 Errors; 0 Warnings; 0 Updated; 2 Copied (2866B); 0 Deleted (0B)
While backuping src/dir with spaces: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
cp -a src/mmm.bat bak1/mmm.bat
cp -a src/t st.m bak1/t st.m
While backuping src: 0 Errors; 0 Warnings; 0 Updated; 4 Copied (5732B); 0 Deleted (0B)
```

Figura 18.3 - Linha de Comandos para Execução do Script.

Neste teste, tendo em conta que a *flag* '-c' foi utilizado e funcionou corretamente, não há necessidade de mostrar a estrutura de *bak1,* já que este não foi criado.

Uso da flag -b

O *tfile.txt* usado para testar esta funcionalidade tinha scripts com caminhos absolutos, relativos, bloqueia o *backup* de *hidden* files e bloqueia também o *backup* de ficheiros com espaços. Continha o seguinte conteúdo:

```
.hidendir/file with spaces.mojo
/home/goncalo/Desktop/SO/projeto1/src/dir with spaces/subdir/.hiden.txt
bat.txt
```

```
 % ./backup_summary.sh -b tfile.txt  src bak1
mkdir bak1
cp -a src/.hiden.bat bak1/.hiden.bat
mkdir bak1/.hidendir
src/.hidendir/file with spaces.mojo ignored
While backuping src/.hidendir: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
src/bat.txt ignored
mkdir bak1/dir with spaces
mkdir bak1/dir with spaces/subdir
src/dir with spaces/subdir/.hiden.txt ignored
cp -a src/dir with spaces/subdir/script.bat bak1/dir with spaces/subdir/script.bat
While backuping src/dir with spaces/subdir: 0 Errors; 0 Warnings; 0 Updated; 1 Copied (1462B); 0 Deleted (0B)
While backuping src/dir with spaces: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
cp -a src/mmm.bat bak1/mmm.bat
cp -a src/t st.m bak1/t st.m
While backuping src: 0 Errors; 0 Warnings; 0 Updated; 3 Copied (4299B); 0 Deleted (0B)
```

Figura 19.1

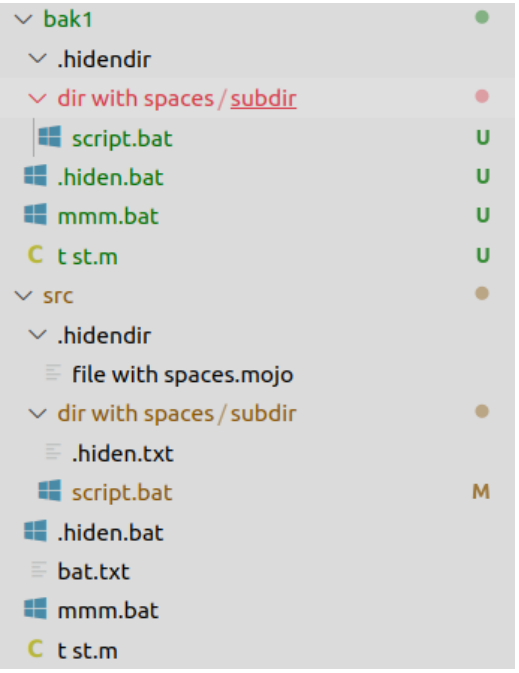

Figura 19.2

Figura 19.1 - Linha de Comando para Execução do *Script*.

Figura 19.2 - Estrutura de ambos *src* e *backupdir* pós execução(Diretoria *bak1* não existia previamente).

Uso da flag -r

Para o teste desta *flag* o padrão *regex* usado foi: '.**hiden.***', ou seja, só os ficheiros que sigam esse padrão serão *backed up.*

```
 % ./backup_summary.sh -r '.*hiden.*'  src bak1
mkdir bak1
cp -a src/.hiden.bat bak1/.hiden.bat
mkdir bak1/.hidendir
While backuping src/.hidendir: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
mkdir bak1/dir with spaces
mkdir bak1/dir with spaces/subdir
cp -a src/dir with spaces/subdir/.hiden.txt bak1/dir with spaces/subdir/.hiden.txt
While backuping src/dir with spaces/subdir: 0 Errors; 0 Warnings; 0 Updated; 1 Copied (1433B); 0 Deleted (0B)
While backuping src/dir with spaces: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
While backuping src: 0 Errors; 0 Warnings; 0 Updated; 1 Copied (1433B); 0 Deleted (0B)
```

Figura 20.1

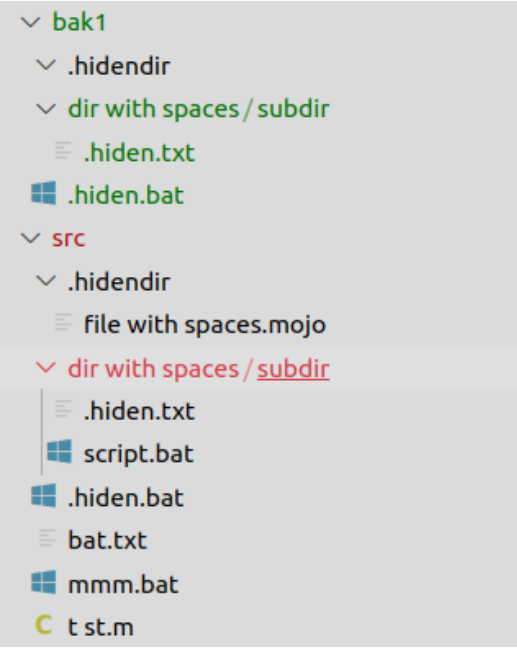

Figura 20.2

Figura 20.1 - Linha de Comandos para Execução do *Script*.

Figura 20.2 - Estrutura de ambos *src* e *bak1* pós execução(Diretoria *bak1* não existia previamente).

3.3 Sumário

Serão dados alguns cenários para testar a funcionalidade do sumário:

Erros e avisos

Para simular uma situação de erro será mudada a permissão de ler, escrever e executar de um dos ficheiros de *src,* não permitindo então o *backup* (*update*) desse ficheiro. Para a situação de aviso, será alterada a *counterpart* de um ficheiro no *bak1,* obtendo um erro do ficheiro de *backup* ser mais recente do que o ficheiro original, quando este estiver para ser atualizado.

```
% chmod a-rwx script.bat
```

Figura 21.1 - Mudança de permissões do script 'script.bat'

```
% ./backup_summary.sh src bak1
While backuping src/.hidendir: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
cp -a src/dir with spaces/subdir/script.bat bak1/dir with spaces/subdir/script.bat
ERROR: couldn't copy src/dir with spaces/subdir/script.bat
While backuping src/dir with spaces/subdir: 1 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
While backuping src/dir with spaces: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
WARNING: backup entry src/mmm.bat is newer than bak1; Should not happen
While backuping src: 0 Errors; 1 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
```

Figura 21.2

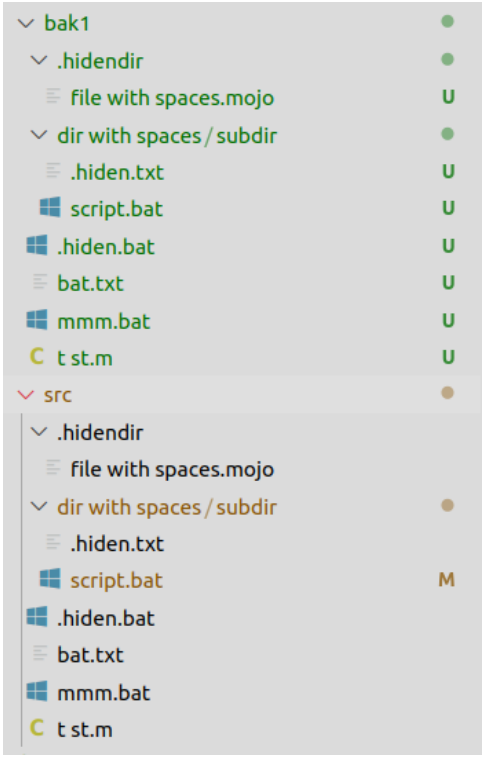

Figura 21.3

Figura 21.2 - Linha de Comando para Execução do *Script*. Apresentando o erro e o aviso mencionado acima.

Figura 21.3 - Estrutura de ambos *src* e *bak1* pós execução(mantém a mesma estrutura, como desejado).

Atualizados, cópias, removidos e tamanho em bytes

Para a simulação destes acontecimentos, é necessário atualizar um ficheiro do diretório de trabalho (atualização, neste caso do ficheiro '*mmm.bat'*) e eliminar do diretório de trabalho um ficheiro (*'bat.txt'*), para que seja apagado a sua *counterpart* no diretório de bak1. É necessário mencionar que estas alterações serão feitas depois do script já ter sido corrido uma vez (onde será testada a capacidade de guardar o número de cópias), sem nenhuma *flag,* nem outro tipo de alterações que comprometam os resultados que pretendemos obter.

```
 % ./backup_summary.sh src bak1
mkdir bak1
cp -a src/.hiden.bat bak1/.hiden.bat
mkdir bak1/.hidendir
cp -a src/.hidendir/file with spaces.mojo bak1/.hidendir/file with spaces.mojo
While backuping src/.hidendir: 0 Errors; 0 Warnings; 0 Updated; 1 Copied (1432B); 0 Deleted (0B)
cp -a src/bat.txt bak1/bat.txt
mkdir bak1/dir with spaces
mkdir bak1/dir with spaces/subdir
cp -a src/dir with spaces/subdir/.hiden.txt bak1/dir with spaces/subdir/.hiden.txt
cp -a src/dir with spaces/subdir/script.bat bak1/dir with spaces/subdir/script.bat
While backuping src/dir with spaces/subdir: 0 Errors; 0 Warnings; 0 Updated; 2 Copied (2895B); 0 Deleted (0B)
While backuping src/dir with spaces: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
cp -a src/mmm.bat bak1/mmm.bat
cp -a src/t st.m bak1/t st.m
While backuping src: 0 Errors; 0 Warnings; 0 Updated; 4 Copied (5732B); 0 Deleted (0B)
```

Figura 22.1 - Execução de Script Inicial

Esta execução inicial demonstra o bom funcionamento do sumário no que toca ao número de ficheiros copiados durante o processo de *backup*.

```
 % ./backup_summary.sh src bak1
While backuping src/.hidendir: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
While backuping src/dir with spaces/subdir: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
While backuping src/dir with spaces: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
cp -a src/mmm.bat bak1/mmm.bat
While backuping src: 0 Errors; 0 Warnings; 1 Updated; 0 Copied (0B); 1 Deleted (1433B)
```

Figura 22.2

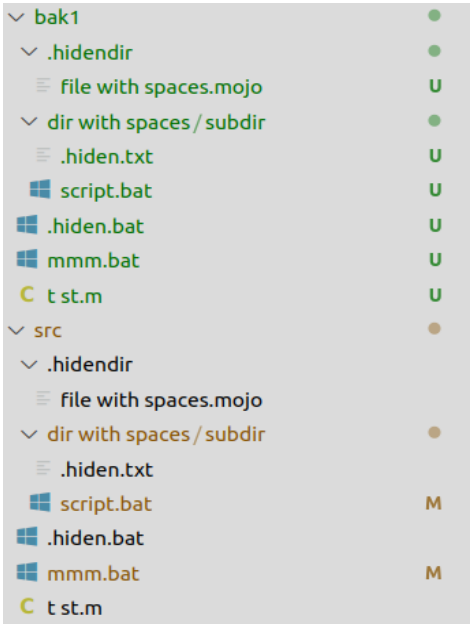

Figura 22.3

Figura 22.2 - Linha de Comandos para Execução do *Script*. Apresentando a atualização e remoção de ficheiros, como mencionado acima.

Figura 22.3 - Estrutura de ambos *src* e *bak1* pós execução.

```
 % stat -c %s bat.txt
1433
```

Figura 22.4 - Tamanho do Ficheiro Removido

Como é possível verificar ao comparar as figuras 22.2 e 22.4, o tamanho do ficheiro removido mencionado na linha de comando coincide com o valor presente em 22.4.

3.4 *backup_check.sh*

Ao usar este script usando como primeiro argumento '*src'* e segundo *'bak1*', logo após o primeiro *backup* ter sido feito, tornando assim impossível que haja algum tipo de diferença entre os ficheiros destes diretórios, obtemos o seguinte output:

```
 % ./backup_check.sh src bak1
goncalo@Swiftie2 ~/Desktop/SO/projeto1
```

Figura 23.1 - Output para backup_check.sh Quando Não Existem Diferenças em src e bak1

Fazendo agora uma alteração no ficheiro '*mmm.bat',* na diretoria *src,* obtemos o seguinte resultado:

```
 % ./backup_check.sh src bak1
src/mmm.bat bak1/mmm.bat differ
```

Figura 23.2 - Output para backup_check.sh Quando Existem Diferenças em src e bak1

Fica então entendido que o programa backup_check.sh está a executar a sua função perfeitamente.

3.5 *Casos Complexos*

Após todos os testes já demonstrados, falta apenas perceber se o código tem a capacidade de aceitar as várias *flags,* e retornar o output esperado. Será usada a expressão *regex* e o *tfile,* usados acima, nos testes anteriores. Observaremos primeiramente a interação de **-b** e **-r** com a *flag* **-c**. De seguida observaremos o output de apenas **-b** juntamente com **-r**, compararemos o output dos dois e, por fim, verificaremos se a estrutura de *bak1 (*a diretoria de *backup)* corresponde com o esperado. Sendo assim, temos o seguinte:

Uso de -c, -b e -r

```
 % ./backup_summary.sh -c -b tfile.txt -r '.*hiden.*' src bak1
mkdir bak1
cp -a src/.hiden.bat bak1/.hiden.bat
mkdir bak1/.hidendir
src/.hidendir/file with spaces.mojo ignored
While backuping src/.hidendir: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
src/bat.txt ignored
mkdir bak1/dir with spaces
mkdir bak1/dir with spaces/subdir
src/dir with spaces/subdir/.hiden.txt ignored
While backuping src/dir with spaces/subdir: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
While backuping src/dir with spaces: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
While backuping src: 0 Errors; 0 Warnings; 0 Updated; 1 Copied (1433B); 0 Deleted (0B)
```

Figura 24.1 - Output com o Uso de Todas as Flags

Uso de -b e -r

```
 % ./backup_summary.sh -b tfile.txt -r '.*hiden.*' src bak1
mkdir bak1
cp -a src/.hiden.bat bak1/.hiden.bat
mkdir bak1/.hidendir
src/.hidendir/file with spaces.mojo ignored
While backuping src/.hidendir: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
src/bat.txt ignored
mkdir bak1/dir with spaces
mkdir bak1/dir with spaces/subdir
src/dir with spaces/subdir/.hiden.txt ignored
While backuping src/dir with spaces/subdir: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
While backuping src/dir with spaces: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
While backuping src: 0 Errors; 0 Warnings; 0 Updated; 1 Copied (1433B); 0 Deleted (0B)
```

Figura 24.2

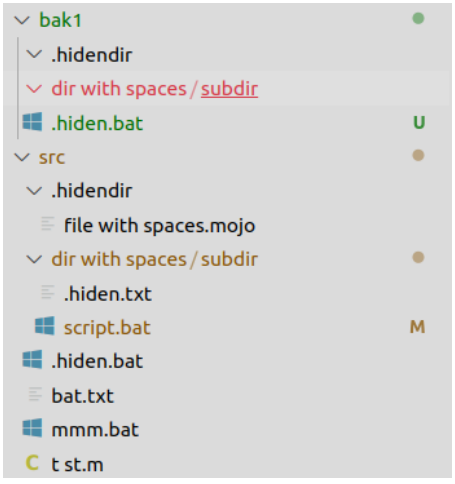

Figura 24.3

Figura 24.2 - Linha de Comando para Execução do *Script*.

Figura 24.3 - Estrutura de ambos *src* e *bak1* pós execução.

Como podemos confirmar ao observar as figuras 24.1 e 24.2, a *flag* **-c** não interfere no *output* para a consola, visto que todas as linhas se mantiveram ao retirar a mencionada *flag*. Observando ainda a figura 24.2, percebemos que foram ignorados os seguintes ficheiros:

```
.hidendir/file with spaces.mojo
/home/goncalo/Desktop/SO/projeto1/src/dir with spaces/subdir/.hiden.txt
bat.txt
```

Que coincidem com os ficheiros a serem ignorados pela *flag* **-b**, inseridos em *tfile*. Percebe-se ainda que, ao observar a figura 24.3, só foram copiados ficheiros ou diretorias que contêm ficheiros, que correspondem ao padrão regex '.*hiden.*'.

Verificamos então que é possível usar todas as *flags* em simultâneo e obter os resultados esperados, sem comprometer o processo de *backup*.

4. Conclusão

Concluímos assim este projeto, respeitando os requisitos descritos no guião do mesmo, e seguindo os exemplos de texto fornecidos pelo guião, o que nos orientou para um desenvolvimento consistente e minimamente robusto. Durante a sua execução, consolidamos os conceitos introdutórios da organização dum sistema operativo, aprendidos nas aulas práticas e teóricas, e como usar as ferramentas da *BASH* para desenvolver uma ferramenta com casos de uso reais. Mas nem tudo foi pêra doce, uma das maiores barreiras que tivemos de ultrapassar foi aprender e descobrir tantos comandos e utilidades da *BASH* novos(as) e tentar aplicá-los à nossa solução. Uma outra dificuldade foi a sintaxe da própria *BASH*, que pode ser facilmente entendida à primeira vista, mas que revela-se muito difícil de masterizar devido ao seu enorme poder e versatilidade.

5. Bibliografia

1) Bash Manual - www.gnu.org/software/bash/manual/bash.html
2) Bash common pitfalls - mywiki.wooledge.org/BashPitfalls
3) Stackoverflow - pt.stackoverflow.com
4) Bash cheatsheet - https://devhints.io/bash