



universidade
de aveiro

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

Sistemas Operativos Relatório

Professor

José Nuno Panelas Nunes Lau (nunolau@ua.pt)

Departamento de Eletrónica, Telecomunicações e Informática

Filipe Marques
120303

Henrique Lopes
119954

Turma P5

Índice

1. Introdução.....	3
2. Estrutura do código.....	4
2.1 Criação do script backup_files.sh.....	4
2.1.1 Função sincronizar_arquivos().....	4
2.1.2 Função remover_arquivos_inexistentes().....	5
2.1.3 Código principal.....	6
2.2 Criação do script backup.sh.....	7
2.2.1 Função usage().....	7
2.2.2 Função sincronizar_arquivos().....	8
2.2.3 Função remover_arquivos_inexistentes().....	9
2.2.3 Código principal.....	10
2.3 Criação do script backup_summary.sh.....	12
2.3.1 Função exibir_warnings().....	13
2.3.2 Função sincronizar_arquivos().....	14
2.3.3 Função remover_arquivos_inexistentes().....	16
2.4 Criação do script backup_check.sh.....	17
2.4.1 Função check_files().....	17
2.4.2 Código Principal.....	19
3. Validação e Testes.....	20
3.1 Validação de backup_files.sh.....	20
3.2 Validação de backup.sh.....	21
3.3 Validação de backup_summary.sh.....	23
3.4 Validação de backup_check.sh.....	25
4. Problemas / Solução.....	26
4.1 Leitura dos ficheiros a ignorar.....	26
4.2 Método de leitura de argumentos.....	26
4.3 Expansão da wildcard '*'.....	27
5. Conclusão.....	27
6. Bibliografia.....	28

1. Introdução

Este relatório apresenta o desenvolvimento de um projeto cujo objetivo principal foi a criação de uma ferramenta automatizada para gestão de cópias de segurança utilizando scripts em Bash. O trabalho consistiu em implementar um conjunto de funcionalidades que permitissem criar e atualizar backups de forma eficiente, garantindo a preservação da integridade dos dados, a detecção de modificações, e a manutenção da estrutura das diretorias e arquivos.

No decorrer deste projeto, foram desenvolvidos quatro scripts distintos, todos baseados na mesma ferramenta de gestão de cópias de segurança, mas com finalidades e funcionalidades específicas. Para a execução de qualquer um dos scripts, é sempre necessário fornecer dois parâmetros: a diretoria de origem, que contém os arquivos e pastas a serem copiados, e a diretoria de destino, responsável por armazenar os backups. Caso a diretoria de destino não exista, os scripts encarregados da cópia criam-na automaticamente antes de proceder com a operação.

A primeira execução do script visa realizar a cópia completa dos ficheiros e diretórios de trabalho para um local de backup. Em execuções subsequentes, a ferramenta identifica e copia apenas os arquivos que foram modificados ou adicionados, removendo aqueles que já não existem no diretório de origem.

É permitido também introduzir parâmetros adicionais `-c`, `-b` e `-r` ao executar o script. O parâmetro `-c` permite visualizar todos os comandos que seriam executados durante o processo de backup, sem que nenhum deles seja efetivamente realizado, oferecendo ao utilizador a possibilidade de verificar previamente o comportamento esperado do script. O parâmetro `-b` exige como argumento um ficheiro de texto contendo uma lista de ficheiros ou diretorias que devem ser excluídos do backup, assegurando que estes não sejam copiados. Já o parâmetro `-r` requer uma expressão regular como argumento, permitindo que apenas os ficheiros ou diretorias que correspondam à expressão sejam incluídos no backup, ignorando tudo o resto.

O relatório detalha a abordagem adotada para a conceção e implementação do projeto, as etapas do desenvolvimento das diversas funcionalidades, os testes realizados para validar o comportamento esperado da solução e os desafios encontrados durante o processo.

2. Estrutura do código

2.1 Criação do script backup_files.sh

Começamos primeiramente por desenvolver o script backup_files.sh para criar e manter um backup incremental de arquivos presentes em uma diretoria de trabalho, garantindo eficiência e integridade de dados. Ele permite copiar apenas os arquivos novos ou modificados, além de oferecer uma opção para simulação das operações.

2.1.1 Função sincronizar_arquivos()

Esta função é responsável por garantir que todos os arquivos novos ou modificados na diretoria de origem sejam copiados para a diretoria de backup.

1. Iteração sobre ficheiros: Utiliza um for loop para percorrer os arquivos diretamente, sem depender de ferramentas externas como find.
2. Caso o ficheiro exista e seja ficheiro regular, manipula-se o valor da variável arquivo para ter o caminho do ficheiro no backup. A linha responsável por isto substitui a diretoria base do backup de um ficheiro pela diretoria base
3. Condição de atualização: Um arquivo será copiado caso:
 - Não exista diretoria de backup.
 - Seja mais recente que o correspondente no backup.
4. Execução condicional: No modo de checking (-c), a operação de cópia não é realizada, mas o comando é exibido no terminal.

```

3  sincronizar_arquivos() {
4      # Loop para iterar sobre cada arquivo na diretoria origem - sem find
5      for arquivo in "$ORIGEMOG"/*; do
6
7          if [ -f "$arquivo" ]; then
8
9              backup="$BACKUPOG${arquivo#$ORIGEMOG}"
10
11              if [ ! -e "$backup" ] || [ "$arquivo" -nt "$backup" ]; then
12                  if [[ "$CHECK" == false ]]; then
13                      cp -a "$arquivo" "$backup"
14                  fi
15                  echo "cp -a ${arquivo#$(dirname "$ORIGEMOG")/" } ${backup#$(dirname "$BACKUPOG")/" }"
16              fi
17          fi
18      done
19  }
--

```

2.1.2 Função remover_arquivos_inexistentes()

Esta função assegura que o backup seja uma réplica precisa da origem, removendo arquivos que não existam da diretoria de trabalho.

1. Iteração sobre cada ficheiro na diretoria destino. Utilizamos o mesmo método que usamos na função `sincronizar_arquivos()`.
2. Verifica se o ficheiro existe e é um ficheiro regular e depois manipula-se o valor da variável `arquivo` para ter o caminho do ficheiro no backup da mesma forma que anteriormente.
3. Caso o ficheiro origem não exista e o modo checking esteja desativado, remove o ficheiro na diretoria destino. De qualquer forma, irá sempre imprimir o comando.

```

21  remover_arquivos_inexistentes() {
22      # Loop para iterar sobre cada ficheiro na diretoria de backup
23      for arquivo in "$BACKUPOG"/*; do
24
25          if [ -f "$arquivo" ]; then
26
27              origem="$ORIGEMOG/${arquivo#$BACKUPOG}"
28
29              if [ ! -e "$origem" ]; then
30                  if [[ "$CHECK" == false ]]; then
31                      rm "$arquivo"
32                  fi
33                  echo "rm ${arquivo#$(dirname "$BACKUPOG")/" }"
34              fi
35          fi
36      done
--

```

2.1.3 Código principal

Aqui vamos analisar todas as verificações feitas antes de ser chamadas as funções e fazer o backup

1. Se forem introduzidos menos que dois argumentos ou mais de 3, imprime uma mensagem a mostrar o uso do comando e sai do programa.
2. Inicia a variável booleana responsável pelo modo checking como falsa. Na verificação dos argumentos, caso tenham sido introduzidos 3 argumentos e o primeiro for '-c', muda o valor da variável para verdadeiro e atribui os argumentos que representam as diretorias origem e destino às respectivas variáveis.
3. Se existirem 3 argumentos mas o primeiro não é '-c' imprime uma mensagem a mostrar o uso do comando e sai do programa.
4. Caso contrário, apenas atribui os argumentos que representam as diretorias origem e destino às respectivas variáveis.

```

47  if [ $# -lt 2 ] || [ $# -gt 3 ]; then
48      echo "Usage: $0 [-c] <source.directory> <backup.directory>"
49      exit 1
50  fi
51
52  CHECK=false
53  if [[ $# -eq 3 ]] && [[ "$1" == "-c" ]]; then
54      CHECK=true
55      ORIGEMOG="$2"
56      BACKUPOG="$3"
57
58  elif [[ $# -eq 3 ]] && [[ "$1" != "-c" ]]; then
59      echo "Usage: $0 [-c] <source.directory> <backup.directory>"
60      exit 1
61
62  else
63      ORIGEMOG="$1"
64      BACKUPOG="$2"
65  fi
66

```

5. Após verificar e atribuir os argumentos às respectivas variáveis, verifica se o argumento passado a 'ORIGEMOG' não é uma diretoria e não existe. Caso esta condição se verifique, sai do programa.
6. Depois verifica se o argumento passado a 'BACKUPOG' não é uma diretoria e não existe. Caso esta condição se verifique e o modo checking esteja desativado, cria a diretoria backup e mostra o comando no terminal.

7. Verificamos de seguida se a diretoria destino não tem permissões para escrever ou a diretoria origem não tem permissões para ler e o modo checking está desativado. Caso isto se verifique, sai do programa pelo erro 2 pois não é problema de argumentos.
8. Por fim, chama-se a função que sincroniza os ficheiros e caso a diretoria backup exista, chama-se também a função responsável por remover os ficheiros inexistentes.

```

67  if [ ! -d "$ORIGEMOG" ]; then
68      exit 1
69  fi
70
71  if [ ! -d "$BACKUPOG" ]; then
72      if [[ "$CHECK" == false ]]; then
73          mkdir -p "$BACKUPOG"
74      fi
75      echo "mkdir -p ${BACKUPOG#$(dirname "$BACKUPOG")}/"
76  fi
77
78  if ([ ! -w "$BACKUPOG" ] || [ ! -r "$ORIGEMOG" ]) && [[ $CHECK == false ]]; then
79      exit 2
80  fi
81
82  sincronizar_arquivos
83
84  if [[ -d "$BACKUPOG" ]]; then
85      remover_arquivos_inexistentes
86  fi

```

2.2 Criação do script backup.sh

Para este script, fizemos tudo o que foi feito no `backup_files()` mas tivemos em consideração os 2 parâmetros adicionais `-b` e `-r`. Tivemos que alterar algum código de forma a conseguir integrar estes parâmetros e tivemos que usar outro método para a introdução de argumentos.

2.2.1 Função `usage()`

Esta é uma simples função responsável por imprimir o como usar o comando para executar o script e sair do programa pelo erro 1 que no nosso código identifica erros nos argumentos.

```

1  #!/bin/bash
2
3  usage() {
4      echo "Usage: $0 [-c] [-b tfile] [-r regexpr] <source_directory> <backup_directory>"
5      exit 1
6  }
7

```

2.2.2 Função sincronizar_arquivos()

Visto que utilizamos a mesma função que no script anterior, vamos apenas mencionar as alterações feitas.

1. A primeira grande diferença é que neste script, para chamar a função, tivemos em consideração a passagem de argumentos. Dado isto, atribuímos imediatamente o 4º e o 5º argumento às variáveis responsáveis por guardar a diretoria origem e destino.
2. Dado que agora devemos fazer a cópia das diretorias também, iteramos por todos os itens existentes na diretoria origem e depois manipulamos o nome do item para criar o caminho do item no backup. Também manipulamos o nome do item para ter apenas o nome da base, excluindo o resto do caminho.
3. Para o argumento ‘-b’, iteramos sobre todos os itens na lista de ficheiros a excluir e, se o nome do item for igual ao ficheiro a excluir atual, salta dois loops para fora, não fazendo a cópia desse item.
4. Verifica se a variável que guarda a expressão regular está definida e se o item não corresponder à expressão salta fora do loop e ignora esse item. O código ‘grep -qE “\$REGEX”’ procura uma combinação para usar como expressão regular estendida sem produzir output.

```

8  sincronizar_arquivos() {
9      local ORIGEM="$4"
10     local BACKUP="$5"
11
12     for item in $ORIGEM/*; do
13
14         nome_item=$(basename "$item")
15         backup="$BACKUP${item#$ORIGEM}"
16
17         for exclude in "${excluded_files[@]}"
18         do
19             if [[ "$exclude" == "$nome_item" ]]
20             then
21                 continue 2
22             fi
23         done
24
25         # Verifica a expressão regular se estiver definida
26         if [[ -n "$REGEX" ]] && ! echo "$nome_item" | grep -qE "$REGEX"; then
27             continue
28         fi
29     done
30 }
```

5. Se o item em questão for uma diretoria e não existir no backup, então cria a diretoria se o modo checking estiver desativado. De qualquer das formas, para todas as diretorias vai ser chamada a função recursivamente para fazer o backup do conteúdo dentro das subdiretorias

6. Caso o item seja um ficheiro, se este não existir no backup ou for mais recente que o seu backup, é feita a cópia do item, se o modo checking estiver desativado.

```

31     if [[ -d "$item" ]]
32     then
33         if [[ ! -d $backup ]]
34         then
35             if [[ "$CHECK" == false ]]
36             then
37                 mkdir $backup
38             fi
39             echo "mkdir ${backup#$(dirname $BACKUP0G)/}"
40         fi
41
42         sincronizar_arquivos "$CHECK" "$EXCLUDE_LIST" "$REGEX" "$item" "$backup"
43
44     elif [[ -f "$item" ]]
45     then
46         if [ ! -e "$backup" ] || [ "$item" -nt "$backup" ]
47         then
48             if [[ "$CHECK" == false ]]
49             then
50                 cp -a "$item" "$backup"
51             fi
52             echo "cp -a ${item#$(dirname $ORIGEM0G)/} ${backup#$(dirname $BACKUP0G)/}"
53         fi
54     fi
55 done
56 }

```

2.2.3 Função remover_arquivos_inexistentes()

Tal como fizemos na função responsável por sincronizar os ficheiros, baseamo-nos na função `remover_arquivos_inexistentes()` feita anteriormente, alterando apenas o necessário

1. Começamos por atribuir os caminhos às respectivas variáveis, para cada item no backup, manipulamos o valor do item para ter acesso ao caminho de item na origem. Fazemos uma pequena verificação no início do loop para verificar se o item é igual ao backup para passar à frente da própria diretoria do backup e evitar qualquer erro.
2. Se o item for uma diretoria e não existir, se o modo checking estiver desativado, remove a diretoria e todos os seus conteúdos e salta fora do loop dessa diretoria. Caso contrário, chama-se recursivamente a função para verificar se há ficheiros ou sub-diretorias a remover dentro da diretoria atual.
3. Caso o item seja um ficheiro, apenas se verifica se ele não está na origem e se o modo checking estiver desativado ele remove-o do backup.

```

58 remover_arquivos_inexistentes() {
59     local ORIGEM="$2"
60     local BACKUP="$3"
61
62     for item in "$BACKUP"/*
63     do
64         if [[ "$item" == "$BACKUP" ]]
65         then
66             continue
67         fi
68
69         origem="$ORIGEM/${item#$BACKUP/}"
70
71         if [[ -d "$item" ]]
72         then
73             if [[ ! -d "$origem" ]]
74             then
75                 if [[ "$CHECK" == false ]]
76                 then
77                     rm -rf "$item"
78                 fi
79             fi
80             continue
81         fi
82     done
83
84     remover_arquivos_inexistentes "$CHECK" "$origem" "$item"
85
86     elif [[ -f "$item" ]]
87     then
88         if [[ ! -f "$origem" ]]
89         then
90             if [[ "$CHECK" == false ]]
91             then
92                 rm "$item"
93             fi
94         fi
95     fi
96 done
97 }

```

2.2.3 Código principal

Aqui usámos outro método para a leitura dos argumentos um pouco mais complexo, mas mais prático tendo em conta que alguns argumentos são opcionais e uns levam argumentos.

1. Começamos por inicializar as variáveis responsáveis por guardar os argumentos, que neste caso são o modo checking, o caminho da lista de ficheiros a ignorar e a expressão regular. E depois inicializamos uma lista vazia criada para guardar os nomes dos ficheiros que devem ser ignorados.
2. Depois utilizamos a utilidade ‘getopts’ para validar os argumentos que vai correr todos os argumentos opcionais e executar o respetivo bloco se encontrar o argumento. O uso desta utilidade está explicado melhor no tópico de problemas/soluções.

3. A seguir à leitura dos argumentos adicionais, se os argumentos obrigatórios forem diferentes a 2 é chamada a função usage. Caso contrário, atribui os valores às respectivas variáveis.
4. Passando à frente as várias verificações que já foram explicadas no script anterior, adicionamos uma nova verificação para saber se a lista de ficheiros a ignorar não está vazia. Se não está vazia, itera sobre cada linha do ficheiro e guarda os valores na lista sem espaços brancos no início e final da linha.
5. Por fim chamamos a função com os argumentos para começar a cópia de segurança e se existir backup chamamos também a função de remover arquivos.

```
125 if [ ! -d "$ORIGEMOG" ]
126 then
127     echo "$1 is not a directory"
128     exit 1
129 fi
130
131 if [ ! -d "$BACKUPOG" ]
132 then
133     if [[ "$CHECK" == false ]]
134     then
135         mkdir -p "$BACKUPOG"
136     fi
137     echo "mkdir ${BACKUPOG#"$(dirname $BACKUPOG)"/}"
138 fi
139
140
141 if ([ ! -w "$BACKUPOG" ] || [ ! -r "$ORIGEMOG" ]) && [[ $CHECK == false ]]
142 then
143     echo "Error in permissions"
144     exit 2
145 fi
```

```

100 CHECK=false
101 EXCLUDE_LIST=""
102 REGEX=""
103 excluded_files=()
104
105
106 while getopts "cb:r:" opt
107 do
108     case "$opt" in
109         c) CHECK=true ;;
110         b) EXCLUDE_LIST="$OPTARG" ;;
111         r) REGEX="$OPTARG" ;;
112         *) usage ;;
113     esac
114 done
115
116 shift $((OPTIND - 1))
117
118 if [ $# -ne 2 ]; then
119     usage
120 fi
121
122 ORIGEMOG="$1"
123 BACKUPOG="$2"
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147 if [[ -n "$EXCLUDE_LIST" ]]
148 then
149     while IFS= read -r line || [[ -n "$line" ]];
150     do
151         line=$(echo "$line" | xargs)
152         excluded_files+=("$line")
153     done < "$EXCLUDE_LIST"
154 fi
155
156 sincronizar_arquivos "$CHECK" "$EXCLUDE_LIST" "$REGEX" "$ORIGEMOG" "$BACKUPOG"
157 if [[ -e "$BACKUPOG" ]]
158 then
159     remover_arquivos_inexistentes "$CHECK" "$ORIGEMOG" "$BACKUPOG"
160 fi
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

2.3 Criação do script backup_summary.sh

O script `backup_summary.sh` foi desenvolvido como uma evolução do `backup.sh`, incorporando uma funcionalidade adicional para fornecer um resumo detalhado de cada diretório processado durante o backup. Este resumo apresenta informações como o número de erros, avisos, arquivos atualizados, copiados e excluídos, bem como o tamanho total dos arquivos manipulados. Essa funcionalidade foi adicionada para tornar o processo de backup mais transparente e fácil de monitorar, permitindo ao usuário identificar rapidamente quaisquer problemas ou inconsistências.

Claro que na implementação do `backup_summary.sh` mantivemos grande parte da lógica do `backup.sh`, mas introduzimos, como dito em cima, modificações para calcular e exibir o resumo no final do processamento de cada diretório. As principais alterações e funcionalidades implementadas são descritas a seguir.

2.3.1 Função `exibir_warnings()`

Uma das adições mais importantes que realizamos neste script foi a função `exibir_warnings()`, que calcula e exibe o número de erros, avisos, arquivos atualizados, copiados e excluídos, bem como o tamanho total dos arquivos manipulados.

Esta função recebe um argumento que é guardado na variável local `“dir_path”`. Declara uma variável local `“relative_path”` com o valor de `“dir_path”`, removendo o prefixo `‘ORIGEMOG’` se ele estiver presente. Isso é feito usando `‘${dir_path#$ORIGEMOG}’`. De seguida construímos o caminho relativo atribuindo o nome base de `‘ORIGEMOG’` concatenado com o valor atual de `‘relative_path’` `‘(basename "$ORIGEMOG")’` retorna apenas o nome do diretório base de `‘ORIGEMOG’`, sem o caminho completo).

Para conseguirmos ter o output desejado, tivemos ainda que remover uma barra inicial (`/`) de `“relative_path”`, se houver. Isso é feito usando a expansão de parâmetros `‘${relative_path#/}’`.

```
8  exibir_warnings() {
9      local dir_path="$1"
10     local relative_path="${dir_path#$ORIGEMOG}"
11
12     relative_path=$(basename "$ORIGEMOG")$relative_path
13     relative_path=${relative_path#/}
```

2.3.2 Função sincronizar_arquivos()

A lógica da função `sincronizar_arquivos()` permanece semelhante à do `backup.sh`, mas foi adaptada para calcular métricas ao processar cada arquivo.

1. Começamos por usar variáveis locais (`local`) dentro da função para controlar e registrar o estado do processo de backup. Essas variáveis são essenciais para monitorar erros, contabilizar operações realizadas, e calcular métricas relacionadas ao tamanho dos arquivos manipulados.

```

25     local error_count=0
26     local warning_count=0
27     local update_count=0
28     local copy_count=0
29     local delete_count=0
30     local copied_size=0
31     local deleted_size=0

```

2. A variável “`error_count`” contabiliza o número de erros encontrados durante o processamento, como problemas de permissão de leitura ou escrita.

```

46     if [[ ! -r "$item" ]]; then
47     then
48         echo "ERROR: \"$relative_item\" does not have reading permissions."
49         ((error_count++))
50         continue
51     fi

```

3. A variável “`warning_count`” contabiliza o número de avisos, como quando uma entrada no backup é mais recente que a correspondente no diretório de trabalho.

```

79     elif [[ -f "$item" ]]; then
80         if [[ -e "$backup" ]] && [[ "$backup" -nt "$item" ]]; then
81             echo "WARNING: backup entry $backup is newer than $item; Should not happen"
82             ((warning_count++))
83         fi
84

```

4. A variável “`update_count`” conta o número de arquivos que foram atualizados no backup porque estavam desatualizados em relação ao diretório de trabalho.

```

85     if [[ ! -e "$backup" ]] || [[ "$item" -nt "$backup" ]]; then
86         if [[ ! -e "$backup" ]]; then
87             ((copy_count++))
88             file_size=$(stat -c%s "$item")
89             copied_size=$((copied_size + file_size))
90         else
91             ((update_count++))
92         fi

```

5. A variável “copy_count” conta o número de novos arquivos copiados para o backup.

```

85     if [[ ! -e "$backup" ]] || [[ "$item" -nt "$backup" ]]; then
86         if [[ ! -e "$backup" ]]; then
87             ((copy_count++))

```

6. A variável “copied_size” vai somar o tamanho total dos arquivos/diretórios excluídos do backup, conseguindo assim saber o espaço libertado pelos ficheiros removidos. Para isso foi preciso usarmos o comando “stat” que serve para conseguirmos obter o tamanho do arquivo de origem (\$item) em bytes. A opção “-c%s” especifica que apenas o tamanho do arquivo deve ser retornado.

```

85     if [[ ! -e "$backup" ]] || [[ "$item" -nt "$backup" ]]; then
86         if [[ ! -e "$backup" ]]; then
87             ((copy_count++))
88             file_size=$(stat -c%s "$item")
89             copied_size=$((copied_size + file_size))
90         else
91             ((update_count++))
92         fi

```

Ao declararmos estas variáveis como locais, elas vão consumir memória apenas durante a execução da função sincronizar_arquivos. Isso significa que, ao sair da função, a memória utilizada por essas variáveis é libertada automaticamente, evitando acúmulo desnecessário e garantindo eficiência no uso de recursos.

7. No final da função é chamada a função `exibir_warnings` para mostrar ao utilizador todas as métricas ao processar cada arquivo/diretório.

```

100
101     exibir_warnings "$ORIGEM"
102 }

```

2.3.3 Função remover_arquivos_inexistentes()

Neste script (backup_summary.sh) seguimos a mesma lógica que no script anterior, mas com algumas alterações para calcular métricas ao processar cada arquivo.

1. Acrescentamos variáveis locais “delete_count” e “deleted_size” no início da função para conseguirmos, monitorar erros e o tamanho de ficheiros/diretorias apagadas.

```
112     local delete_count=0
113     local deleted_size=0
```

2. A variável delete_count conta o número de arquivos/diretórios excluídos do backup por deixarem de existir no diretório de trabalho. A variável “copied_size” vai somar o tamanho total dos arquivos copiados para o backup, usando de novo o comando “stat” que serve para conseguirmos obter o tamanho do arquivo de origem (\$item) em bytes, como já explicado anteriormente.

```
116     if [[ -d "$item" ]]; then
117         if [[ ! -d "$origem" ]]; then
118             if [[ "$CHECK" == false ]]; then
119                 rm -rf "$item"
120             fi
121             echo "rm -rf ${item#$(dirname $BACKUPOG)/}"
122             ((delete_count++))
123             deleted_size=$((deleted_size + $(du -sb "$item" | cut -f1)))
124         fi
125         continue
126     fi
127
128     remover_arquivos_inexistentes "$CHECK" "$origem" "$item"
129
130     elif [[ -f "$item" ]]; then
131         if [[ ! -f "$origem" ]]; then
132             file_size=$(stat -c%s "$item")
133             if [[ "$CHECK" == false ]]; then
134                 rm "$item"
135             fi
136         fi
137         ((delete_count++))
138         deleted_size=$((deleted_size + file_size))
139     fi
140 fi
141 done
```


3. Por último, verificamos se a variável `delete_count` é maior que zero, se for, ele executa as seguintes ações:
 - Usa o comando “`dirname`” para obter o diretório `BACKUPOG` e é armazenado na variável “`base_dir`”.
 - Remove o prefixo “`base_dir`” do caminho de `BACKUPOG` e é armazenado na variável “`relative_backup`”.
 - Por último, chama a função `exibir_warnings` para exibir um resumo detalhado das operações de backup.

```

143     if [[ delete_count -gt 0 ]]; then
144         local base_dir=$(dirname "$BACKUPOG")
145         local relative_backup="${BACKUP#$base_dir/}"
146         exibir_warnings "$relative_backup"

```

2.4 Criação do script `backup_check.sh`

O script `backup_check.sh` foi desenvolvido para verificar a integridade do backup de ficheiros. Ele compara os ficheiros existentes na diretoria de origem com os ficheiros correspondentes no diretório de backup e determina se o conteúdo é idêntico. Caso sejam detectadas discrepâncias, o script gera mensagens detalhadas de saída para informar sobre os problemas encontrados.

O foco deste script não é copiar ficheiros ou lidar com ficheiros novos, mas garantir que os ficheiros já existentes no backup são idênticos aos da origem, usando o comando `md5sum` para calcular e comparar os valores de hash.

2.4.1 Função `check_files()`

A função `check_files()` é o núcleo do script. Ela realiza as seguintes operações:

1. Vai iterar sobre todos os itens na origem, percorrendo todos os ficheiros e subdiretórios no diretório origem.
2. De seguida, vai identificar os diretórios, ou seja:
 - Se o item atual for um diretório, verifica se o diretório correspondente existe no backup.

- Caso exista, chama-se a própria função recursivamente para verificar os conteúdos dentro do subdiretório.
 - Caso contrário, é exibida uma mensagem informando que o diretório está ausente no backup.
3. Também vai identificar os ficheiros, ou seja;
- Se o item atual for um ficheiro, verifica-se se o ficheiro correspondente existe no backup.
 - Caso exista, compara-se o hash md5sum dos dois ficheiros.
 - Caso os hashes sejam diferentes, é exibida uma mensagem indicando que os ficheiros diferem.
 - Caso o ficheiro correspondente não exista no backup, é exibida uma mensagem informando a ausência do ficheiro.
4. Mensagens de saída:
- Cada discrepância é reportada com mensagens detalhadas, incluindo os caminhos relativos dos ficheiros e diretórios.

```

8  check_files() {
9      local ORIGEM="$1"
10     local BACKUP="$2"
11
12     for item in "$ORIGEM"/*; do
13
14         nome_item=$(basename "$item")
15         backup="$BACKUP${item#$ORIGEM}"
16
17         relative_backup="${backup#$(dirname $BACKUP)/}"
18         relative_item="${item#$(dirname $ORIGEM)/}"
19
20         if [[ -d "$item" ]]; then
21             if [[ -d "$backup" ]]; then
22
23                 check_files "$item" "$backup"
24
25             else
26
27                 echo "Directory $relative_backup does not exist"
28             fi
29
30         elif [[ -f "$item" ]]; then
31             if [[ -f "$backup" ]]; then
32                 source_check=$(md5sum "$item" | awk '{print $1}')
33                 backup_check=$(md5sum "$backup" | awk '{print $1}')
34
35                 if [[ "$source_check" != "$backup_check" ]]; then
36
37                     echo "File $relative_item and $relative_backup differ"
38                 fi
39             else
40
41                 echo "File $relative_backup does not exist"
42             fi
43         fi
44     done
45 }

```

2.4.2 Código Principal

O código principal inicializa o script, valida os argumentos e chama a função `check_files()`.

1. Em primeiro lugar vai verificar se o número de argumentos é igual a dois e se os diretórios fornecidos existem.
2. Para o script correr como o planeado, fizemos a chamada da função principal. Inicializamos as variáveis 'ORIGEMOG' e 'BACKUPOG' com os diretórios fornecidos e por fim foi feita a chamada da função `check_files()` passando os diretórios.

```
47  if [ $# -ne 2 ]; then
48      usage
49  fi
50
51  ORIGEMOG="$1"
52  BACKUPOG="$2"
53
54
55  if [ ! -d "$ORIGEMOG" ]; then
56      echo "$1 is not a directory"
57      exit 1
58  fi
59
60
61  if [ ! -d "$BACKUPOG" ]; then
62      echo "$2 is not a directory"
63      exit 1
64  fi
65
66  check_files "$ORIGEMOG" "$BACKUPOG"
--
```

3. Validação e Testes

Para garantir que os scripts desenvolvidos funcionam, definimos vários casos de teste que abrangem tanto situações comuns quanto exceções que poderiam ocorrer no uso dos scripts.

Vamos considerar a diretoria origem ‘testes’ e a diretoria destino ‘backup’.

3.1 Validação de backup_files.sh

Este script tem como objetivo principal a realização de cópias de segurança de ficheiros regulares , contando ainda com a possibilidade de utilização do parâmetro opcional ‘-c’.

- Primeira execução do script:

```
● filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ ./backup_files.sh /home/filipe0219/Documents/S0/testes /home/filipe0219/Documents/S0/backup
mkdir -p backup
cp -a testes/11txt backup/11txt
cp -a testes/1txt backup/1txt
○ filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$
```

Após a primeira execução, se repetir o comando sem alterar nada, nada acontece, tal como é suposto.

- Modo checking funcional:

```
● filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ ./backup_files.sh -c /home/filipe0219/Documents/S0/testes /home/filipe0219/Documents/S0/backup
mkdir -p backup
cp -a testes/11txt backup/11txt
cp -a testes/1txt backup/1txt
```

- Alterei o ficheiro ‘1txt’ e removi o ficheiro “11txt”. Também funciona com modo checking:

```
● filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ ./backup_files.sh /home/filipe0219/Documents/S0/testes /home/filipe0219/Documents/S0/backup
cp -a testes/1txt backup/1txt
rm backup/11txt
```

3.2 Validação de backup.sh

O script backup.sh já tem em consideração várias situações dado que agora é possível fazer a cópia de diretorias e ficheiros e é possível incluir os 3 parâmetros opcionais.

```
● filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ ./backup1.sh /home/filipe0219/Documents/S0/testes /home/filipe0219/Documents/S0/backup
mkdir backup
mkdir backup/1subdiretoria
mkdir backup/1subdiretoria/2subdiretoria
mkdir backup/1subdiretoria/2subdiretoria/3subdiretoria
cp -a testes/1subdiretoria/2subdiretoria/3txt backup/1subdiretoria/2subdiretoria/3txt
cp -a testes/1subdiretoria/2txt backup/1subdiretoria/2txt
cp -a testes/1txt backup/1txt
```

- Primeira execução do script

Após a primeira execução, se repetir o comando sem alterar nada, nada acontece, tal como é suposto.

- Modo checking funcional

```
● filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ ./backup1.sh -c /home/filipe0219/Documents/S0/testes /home/filipe0219/Documents/S0/backup
mkdir backup
mkdir backup/1subdiretoria
mkdir backup/1subdiretoria/2subdiretoria
mkdir backup/1subdiretoria/2subdiretoria/3subdiretoria
cp -a testes/1subdiretoria/2subdiretoria/3txt backup/1subdiretoria/2subdiretoria/3txt
cp -a testes/1subdiretoria/2txt backup/1subdiretoria/2txt
cp -a testes/1txt backup/1txt
```

- Removi o ficheiro '2txt' e a diretoria '3subdiretoria'. Alterei '1txt' e '3txt'. Funciona também com o modo checking.

```
● filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ ./backup1.sh /home/filipe0219/Documents/S0/testes /home/filipe0219/Documents/S0/backup
cp -a testes/1subdiretoria/2subdiretoria/3txt backup/1subdiretoria/2subdiretoria/3txt
cp -a testes/1txt backup/1txt
rm -rf backup/1subdiretoria/2subdiretoria/3subdiretoria
rm -rf backup/1subdiretoria/2txt
```

- Na verificação do parâmetro -b, que recebe ficheiros ou diretorias que devem ser ignoradas, passámos um ficheiro de texto que considera várias situações para ver se está a trabalhar de forma prevista. Este é o ficheiro “tfile” e como os ficheiros devem ser interpretados:

```
ign2
    igndir2    a
    igndir
        ign3
    ign1
```

```
ign2      - ignore
igndir2   - dont ignore
igndir    - ignore
ign3      - ignore
ign1      - ignore
```

(ign2 tem espaços brancos à frente)

Foi feito o backup de todos os itens exceto os que deveriam ser ignorados. A diretoria “igndir2” não foi ignorada, devido à presença de um caractere adicional na

```
● filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ ./backup1.sh -b /home/filipe0219/Documents/S0/tfile /home/filipe0219/Documents/S0/testes /home/filipe0219/Documents/S0/backup
mkdir backup
mkdir backup/1subdiretoria
mkdir backup/1subdiretoria/2subdiretoria
mkdir backup/1subdiretoria/2subdiretoria/3subdiretoria
cp -a testes/1subdiretoria/2subdiretoria/3txt backup/1subdiretoria/2subdiretoria/3txt
cp -a testes/1subdiretoria/2txt backup/1subdiretoria/2txt
mkdir backup/1subdiretoria/igndir2
cp -a testes/1txt backup/1txt
```

mesma linha do ficheiro de configuração, tal como esperado. Também funciona com modo checking ativado.

- Na verificação do parâmetro -c, que indica que, apenas ficheiros que combinam com a expressão regular passada devem ser copiados, passámos a expressão `^regex.*`, que deve aceitar todos os ficheiros ou diretorias que comecem com `regex`. Na diretoria origem `testes` temos a seguinte estrutura :

- testes/
 - |— 1subdiretoria/
 - | |— (conteúdo aleatório...)
 - |— regexdir/
 - | |— regexdir2/
 - | |— regex
 - |— 1txt

Foi feito o backup de todos os ficheiros que correspondem à expressão regular passada, ignorando tudo o resto. Também funciona com modo checking ativado.

```

● filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ ./backup1.sh -r '^regex.
*' /home/filipe0219/Documents/S0/testes /home/filipe0219/Documents/S0/backup
mkdir backup
mkdir backup/regexdir
cp -a testes/regexdir/regex backup/regexdir/regex
mkdir backup/regexdir/regexdir2

```

3.3 Validação de backup_summary.sh

Este script deve fazer o mesmo que o backup.sh faz, mas deve mostrar avisos para certas situações e após o backup de cada diretoria, deve fazer um sumário do que aconteceu.

Este script também permite usar os parâmetros adicionais, mas como já foi demonstrado no script anterior, não voltamos a mostrar nesta validação. Mas foi devidamente testado

- Primeira execução do script:

```

● filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ ./backup_summary.sh /home/filipe0219/Documents/S0/testes /home/filipe0219/Documents/S0/backup
mkdir backup
mkdir backup/1subdiretoria
mkdir backup/1subdiretoria/2subdiretoria
While backuping testes/1subdiretoria/2subdiretoria: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
cp -a testes/1subdiretoria/2txt backup/1subdiretoria/2txt
While backuping testes/1subdiretoria: 0 Errors; 0 Warnings; 0 Updated; 1 Copied (23B); 0 Deleted (0B)
cp -a testes/1txt backup/1txt
While backuping testes: 0 Errors; 0 Warnings; 0 Updated; 1 Copied (6B); 0 Deleted (0B)
    
```

- Modo checking funcional

```

● filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ ./backup_summary.sh -c /home/filipe0219/Documents/S0/testes /home/filipe0219/Documents/S0/backup
mkdir backup
mkdir backup/1subdiretoria
mkdir backup/1subdiretoria/2subdiretoria
While backuping testes/1subdiretoria/2subdiretoria: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
cp -a testes/1subdiretoria/2txt backup/1subdiretoria/2txt
While backuping testes/1subdiretoria: 0 Errors; 0 Warnings; 0 Updated; 1 Copied (23B); 0 Deleted (0B)
cp -a testes/1txt backup/1txt
While backuping testes: 0 Errors; 0 Warnings; 0 Updated; 1 Copied (6B); 0 Deleted (0B)
    
```

- Para o mesmo backup, se alterar algo no diretoria backup, esse item será mais recente que o respetivo item na origem, logo o script deve apresentar um aviso.

```

● filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ ./backup_summary.sh /home/filipe0219/Documents/S0/testes /home/filipe0219/Documents/S0/backup
While backuping testes/1subdiretoria/2subdiretoria: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
While backuping testes/1subdiretoria: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
WARNING: backup entry /home/filipe0219/Documents/S0/backup/1txt is newer than /home/filipe0219/Documents/S0/testes/1txt; Should not happen
While backuping testes: 0 Errors; 1 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
    
```

- Um dos erros que consideramos foi se um ficheiro no backup (considerando que já foi feito pelo menos um backup) não tem permissões de escrita. Deve considerar o erro e mostrar no terminal.

```

● filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ chmod -w /home/filipe0219/Documents/S0/backup/1txt
● filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ ./backup_summary.sh -c /home/filipe0219/Documents/S0/testes /home/filipe0219/Documents/S0/backup
While backuping testes/1subdiretoria/2subdiretoria: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
While backuping testes/1subdiretoria: 0 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
ERROR: backup/1txt does not have writing permissions.
While backuping testes: 1 Errors; 0 Warnings; 0 Updated; 0 Copied (0B); 0 Deleted (0B)
    
```


3.4 Validação de backup_check.sh

Para testar este script, utilizamos a cópia mostrada na validação do backup.sh. Este script deve apenas verificar se o conteúdo num ficheiro que esteja na diretoria origem e na diretoria backup é igual em ambas diretorias.

- Primeira execução após fazer o backup não aparece nada, tal como esperado.

```
● filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ ./backup_check.sh /home/filipe0219/Documents/S0/testes /home/filipe0219/Documents/S0/backup
○ filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$
```

- Se alterar uns ficheiros no backup, mostra no terminal quais os ficheiros que diferem.

```
● filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ ./backup_check.sh /home/filipe0219/Documents/S0/testes /home/filipe0219/Documents/S0/backup
File testes/1subdiretoria/2subdiretoria/3txt and backup/1subdiretoria/2subdiretoria/3txt differ
File testes/1txt and backup/1txt differ
○ filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$
```

- Se alterar uns ficheiros na origem, mostra no terminal quais os ficheiros que diferem.

```
● filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ ./backup_check.sh /home/filipe0219/Documents/S0/testes /home/filipe0219/Documents/S0/backup
File testes/1subdiretoria/2subdiretoria/3txt and backup/1subdiretoria/2subdiretoria/3txt differ
File testes/1subdiretoria/2txt and backup/1subdiretoria/2txt differ
○ filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$
```

- Se o ficheiro diretoria ou o backup não existir, mostra uma mensagem de erro.

```
✖ filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ ./backup_check.sh /home/filipe0219/Documents/S0/non /home/filipe0219/Documents/S0/backup
/home/filipe0219/Documents/S0/non is not a directory
✖ filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$ ./backup_check.sh /home/filipe0219/Documents/S0/testes /home/filipe0219/Documents/S0/non
/home/filipe0219/Documents/S0/non is not a directory
○ filipe0219@filipe0219-L0Q-15APH8:~/Documents/S0/project01_so$
```

4. Problemas / Solução

4.1 Leitura dos ficheiros a ignorar

Problema:

Durante o desenvolvimento do script `'backup.sh'`, mais especificamente no parâmetro adicional que recebe uma lista de ficheiros ou diretórios a serem ignorados, notámos que, caso existissem espaços em branco antes ou depois do nome do ficheiro no ficheiro de texto, o programa interpretava esses espaços em branco como parte do nome do ficheiro. Como resultado, o ficheiro não era corretamente identificado, e o programa acabava por incluí-lo na cópia.

Solução:

No `'for loop'` em que lemos cada linha do ficheiro contendo os nomes dos ficheiros a serem ignorados, utilizamos `'IFS='` para considerar espaços em branco entre os nomes. No entanto, isso faz com que o programa também interprete espaços em branco no início e no final do nome do ficheiro. Para resolver este problema, utilizamos o comando `'xargs'`, que remove os espaços em branco no início e no fim de cada linha.

Desta forma, o programa consegue processar um ficheiro de texto onde os nomes possam conter espaços entre si, mas quaisquer espaços antes ou depois de cada nome não serão considerados.

4.2 Método de leitura de argumentos

Problema:

Como ao executar o script, é possível adicionar flags opcionais onde umas recebem argumentos e outras não, isto torna a leitura de argumentos e flags uma tarefa complicada no projeto.

Solução:

Nós resolvemos isso usando a utilidade `'getopts'`, que facilita a interpretação de flags e argumentos na linha de comandos. O `getopts` é usado para processar as opções fornecidas a um script de forma ordenada. A string `":cb:r:"` especifica as opções permitidas:

- `'c'` é uma opção sem argumento;
- `'b:'` requer argumento (o `:` após o `b` indica que é necessário um argumento);
- `'r:'` da mesma forma que `'b'` requer argumento;
- `':'` no início da string é convenção para lidar com erros em bash de forma efetiva.

`'opt'` é a variável onde é armazenada a letra a ser processada no loop. `'OPTARG'` é uma variável especial que recebe o argumento da opção atual.

Enquanto iterar sobre cada opção, se esta opção for encontrada, executa o respetivo bloco de código à frente.

‘OPTIND’ é uma variável especial do ‘getopts’ que aponta para o índice do próximo argumento a ser processado na linha de comando. Após o loop faz-se ‘shift \$((OPTIND - 1))’ para remover os argumentos já processados, deixando apenas os argumentos restantes que irão ser tratados separadamente no script

4.3 Expansão da wildcard ‘*’

Problema:

Ao iterar sobre os itens numa diretoria, em qualquer script, era detetado um ‘*’ nas diretorias que se encontrassem vazias.

Solução:

Isto acontece porque, quando utilizamos ‘/*’, o shell tenta expandir o ‘*’ para corresponder a todo o conteúdo da diretoria a ser iterada. Se a diretoria estiver vazia, a expansão falha, e a string literal \$ORIGEM/*. Para resolver isto fizemos uma simples verificação após o início do loop para ver se o item atual é igual a “\$ORIGEM”/* de forma a ignorar a wildcard ‘*’.

5. Conclusão

De forma geral, os scripts desenvolvidos neste projeto são ferramentas eficazes para lidar com a gestão de cópias de segurança de diretorias. Eles permitem o utilizador escolher os ficheiros que pretende ou não fazer backup, analisar se os conteúdos foram passados corretamente e copiar apenas ficheiros que sigam um padrão.

Concluimos este projeto de acordo com o que foi proposto, seguindo os parâmetros e requisitos estabelecidos. Os resultados obtidos com a execução dos scripts estão em conformidade com as diretrizes apresentadas no guião.

6. Bibliografia

1. Documentação do Bash:
 - GNU Bash Reference Manual: <https://www.gnu.org/software/bash/manual/>
 - Explicações detalhadas sobre o funcionamento do Bash, incluindo estruturas de controle, funções e manipulação de arquivos.
2. Documentação do md5sum:
 - Linux Man Pages: <https://man7.org/linux/man-pages/man1/md5sum.1.html>
 - Referência para o uso do comando md5sum para verificar integridade de arquivos.
3. Tutoriais sobre Expressões Regulares:
 - Regular-Expressions.info: <https://www.regular-expressions.info/>
 - Guia abrangente para aprender e aplicar expressões regulares em diferentes linguagens, incluindo Bash.
4. Comandos Linux para Manipulação de Arquivos e Diretórios:
 - Linuxize: <https://linuxize.com/post/basic-linux-commands/>
 - Explicações práticas sobre comandos como cp, rm, du, stat, entre outros, usados frequentemente em scripts de backup.
5. Estilo e Boas Práticas em Bash:
 - ShellCheck: <https://www.shellcheck.net/>
 - Ferramenta online para verificar e corrigir problemas de estilo e bugs em scripts Bash.