



deti

universidade de aveiro
departamento de electrónica,
telecomunicações e informática

SISTEMAS OPERATIVOS

Trabalho 1

Ferramenta de criação/atualização de cópias de segurança em bash

Trabalho realizado por:

Diego Aguilar Leon - nº 117122

Dinis Sousa - nº119500

Ano Letivo 2024/2025

Índice

| | |
|------------------------------------------------------|-----------|
| INTRODUÇÃO | 3 |
| ABORDAGEM | 4 |
| 1) BACKUP_FILES.SH | 4 |
| • <i>Verificação de argumentos.....</i> | <i>4</i> |
| • <i>Modo_verificar</i> | <i>4</i> |
| • <i>Copia y remoção de arquivos.....</i> | <i>5</i> |
| 2) BACKUP.SH | 6 |
| • <i>Parâmetros e Flags de Execução.....</i> | <i>6</i> |
| • <i>Copia y remoção de arquivos.....</i> | <i>6</i> |
| 3) BACKUP_SUMMARY.SH | 7 |
| • <i>Definição de Variáveis para Contadores.....</i> | <i>7</i> |
| • <i>Copia y remoção de arquivos.....</i> | <i>8</i> |
| VALIDAÇÃO DA SOLUÇÃO..... | 10 |
| 1) BACKUP_FILES.SH | 10 |
| • <i>Testes.....</i> | <i>10</i> |
| • <i>Erros.....</i> | <i>11</i> |
| 2) BACKUP.SH | 12 |
| • <i>Testes.....</i> | <i>12</i> |
| • <i>Erros.....</i> | <i>14</i> |
| 3) BACKUP_SUMMARY.SH | 14 |
| • <i>Erros.....</i> | <i>15</i> |
| 4) BACKUP_CHECK.SH | 16 |
| • <i>Testes.....</i> | <i>17</i> |

Introdução

O objetivo deste projeto é desenvolver um programa de backup automatizado em bash para criar e atualizar cópias de backup de diferentes diretórios. O script copiará todos os ficheiros e subdiretórios de um ficheiro de origem para um ficheiro de cópia de segurança.

O sistema foi implementado por partes, passámos a lidar com ficheiros simples, considerando apenas as atualizações recentes de ficheiros. No passo seguinte, foi incluída uma estrutura completa de subdiretórios, acrescentando a capacidade de fazer backups de toda a estrutura de diretórios. Posteriormente, foi implementada uma funcionalidade adicional que fornece um resumo detalhado com a contagem de erros, atualizações, cópias e exclusões efetuadas no momento do backup.

Este projeto oferece uma solução prática e eficiente para manter a segurança dos dados e a consistência entre a origem e o destino do backup, otimizando o processo de sincronização e reduzindo o tempo de execução dos backups incrementais.

Abordagem

Conforme indicado no guia, o projeto foi desenvolvido em diferentes fases, criando diferentes versões do programa, começando por uma com código mais simples e com funções mais limitadas, até chegar à versão final do programa que se baseia nos programas anteriores, desenvolvidas e acrescentando novas funções, cada uma das diferentes versões desenvolvidas será descrita com mais detalhes nos pontos seguintes bem como os testes que foram realizados para verificar a correta funcionalidade do código.

1) backup_files.sh

A primeira versão do programa contém o código mais simples para realizar o backup, com funções para copiar e eliminar ficheiros de um diretório.

- Verificação de argumentos

Para que o script funcione corretamente, são necessários pelo menos dois argumentos: diretório de trabalho (origem) e diretório de cópia de segurança (destino).

A condicional if é aplicada para verificar se este foi cumprido, caso contrário será apresentada uma mensagem de erro e o programa terminará a execução.

```
#verificacoes
if [[ "$#" -lt 2 ]]; then #numero argumentos
    printf "Por favor escreva: %s [-c] dir_trabalho dir_backup\n" "$0"
    exit 1
fi
```

Figure 1/ Verificar número de argumentos

- Modo_verificar

Este é inicializado com um valor falso, que será alterado (passando para verdadeiro) se -c for fornecido como argumento. Quando este modo estiver ativado, o script funcionará em modo de simulação, resultando apenas nas ações que serão executadas, sem modificar qualquer ficheiro ou diretório.

Esta função é útil para testar o correto funcionamento do script e realizar testes para as novas funções de futuras versões do código.

É também realizada uma verificação dos argumentos se for fornecido algo diferente de -c, emitindo uma mensagem de aviso de erro. Nesta parte do código também é aplicada uma condicional para verificar a existência do diretório do qual deseja fazer o backup.

```

MODO_VERIFICAR="false"
while getopts ":c" opt; do
    case $opt in
        c) MODO_VERIFICAR=true ;;
        \?) printf "Opção inválida: %s\n" "$OPTARG" >&2; uso ;;
        :) printf "Opção %s requer um argumento\n" "$OPTARG" >&2; uso ;;
    esac
done
shift $((OPTIND - 1))

```

Figure 2/ Modo Verificar

```

if [[ ! -d "$1" ]]; then # existencia da diretoria especificada
    printf "Erro: diretoria '%s' não existe.\n" "$1"
    exit 1
fi

```

Figure 3/ Existência do diretório

- Cópia y remoção de arquivos

A função copia() foi criada para poder fazer o backup dos ficheiros dentro do diretório de origem. Se o arquivo de destino não existe ou é mais antigo que o arquivo de origem, o script copia o arquivo. Em modo de verificação, ele apenas exibe o comando cp, sem realizar a cópia. Caso o diretório de destino não exista, o script o cria usando mkdir -p.

Através de um loop, a função acima descrita é aplicada a todos os ficheiros do diretório de origem, enviando-os para o diretório de backup.

Um segundo loop é aplicado para verificar os ficheiros copiados, se um arquivo no backup não tem uma contraparte no diretório de trabalho, o script o remove, considerando-o obsoleto. Em modo de verificação, ele exibe o comando rm sem realmente remover o arquivo.

```

#copiar os arquivos
copia() {
    fonte="$1"
    destino="$2"
    #echo $destino
    #echo $fonte
    if [[ -d "$destino" || "$fonte" -nt "$destino" ]]; then
        if [[ "$MODO_VERIFICAR" == true ]]; then
            printf "cp -a '%s' '%s'\n" "$fonte" "$destino"
        elif [[ "$MODO_VERIFICAR" == false ]]; then
            #caso backup nao exista ainda
            if [[ ! -d "$2" ]]; then
                mkdir -p "$(dirname "$destino")"
            fi
            cp -a "$fonte" "$destino" || printf "Erro ao copiar '%s' para '%s'\n" "$fonte" "$destino"
        fi
    fi
}

```

Figure 4/ Função copia

```

#remover lixo
for backup_arquivo in "$2"/*; do
    #echo $destino
    if [[ -f "$backup_arquivo" && ! -f "$1/${basename "$backup_arquivo"}" ]]; then
        if [[ "$MODO_VERIFICAR" == false ]]; then
            rm "$backup_arquivo"
        elif [ "$MODO_VERIFICAR" == true ]; then
            printf "rm '%s'\n" "$backup_arquivo"
        fi
    fi
done

```

Figure 5/ Remover lixo

2) backup.sh

Nesta nova versão do código, a primeira parte do código anterior é utilizada como base, são aumentadas novas funções e melhorado o modo de cópia e remoção de ficheiros, permitindo não só fazer backup de ficheiros, mas também de diretórios e subdiretórios.

- Parâmetros e Flags de Execução

Modo de verificação é mantido (-c), tendo a mesma função da versão anterior. Dois parâmetros adicionais são aumentados.

-r <regex> : O script verifica cada arquivo no diretório de trabalho usando a expressão regular fornecida com -r. Somente os arquivos que correspondem à regex especificada serão copiados para o diretório de backup.

-b <blacklist.txt> : Carrega uma lista de exclusões a partir de um arquivo (blacklist.txt), ignorando arquivos e diretórios listados durante a cópia. Caso seja fornecida blacklist e o ficheiro seja válido, o script irá carregar a informação para um array (EXCLUSOES), posteriormente com a função `esta_na_lista()` é verificado se algum ficheiro ou diretório está dentro da lista, que será ignorado para o processo de backup

```
if [[ -n "$BLACKLIST" && -f "$BLACKLIST" ]]; then
    #echo $BLACKLIST
    mapfile -t EXCLUSOES < "$BLACKLIST"
else
    EXCLUSOES=()
fi
```

Figure 6/ Lista de EXCLUSOES

- Cópia y remoção de arquivos

Para copiar ficheiros e diretórios nesta versão, foi criada a função `copiar_recursiva`, o código da função `copy()` desenvolvida anteriormente é utilizado como referência, a função `esta_na_lista()` é aplicada para realizar a eliminação de ficheiros e é aplicada uma condicional para filtrar os itens que correspondem ao (-r) REGEX fornecido.

```
copiar_recursiva() {
    local fonte="$1"
    local destino="$2"

    #fonte
    for item in "$fonte"/*; do
        local nome_base
        nome_base=$(basename "$item")

        #exclusions
        if esta_na_lista "$nome_base"; then
            continue
        fi

        #verificar se o item corresponde regex
        if [[ -n "$REGEX" && ! "$nome_base" =~ $REGEX ]]; then
            continue
        fi

        #copyign
        if [[ -f "$item" ]]; then
            if [[ ! -f "$destino/$nome_base" || "$item" -nt "$destino/$nome_base" ]]; then
                if [[ "$MODULO_VERIFICAR" == true ]]; then
                    printf "cp -a '%s' '%s'\n" "$item" "$destino/$nome_base"
                else
                    mkdir -p "$destino"
                    cp -p "$item" "$destino/$nome_base" || printf "Erro ao copiar '%s'\n" "$item"
                fi
            fi
        elif [[ -d "$item" ]]; then
            #subd
            copiar_recursiva "$item" "$destino/$nome_base"
        fi
    done
}
```

Figure 7/ Função `copiar_recursiva()`

```

esta_na_lista() {
    local arquivo="$1"
    for excluido in "${EXCLUSOES[@]"; do
        #echo a $arquivo
        #echo o $excluido
        if [[ "$arquivo" == "$excluido" || "$arquivo" == "$excluido/*" ]]; then
            return 0
        fi
    done
    return 1
}

```

Figure 8/ Função esta_na_lista()

Para remover ficheiros e diretórios, foi desenvolvida a função remove_extras(). Se o item não existe mais no diretório de trabalho ou está na lista de exclusões, ele é removido. A função é responsável por verificar se o ficheiro correspondente existe no diretório fonte. A remoção depende do valor da variável MODO_VERIFICAR:

Modo de verificação (MODO_VERIFICAR=true): Mostra uma mensagem indicando que o arquivo seria removido, sem realizar a ação.

Modo de execução (MODO_VERIFICAR=false): Remove o arquivo usando rm "\$backup_arquivo".

Ambas as funções (copiar e remover) funcionam recursivamente para processar subdiretórios e ficheiros dentro das mesmas, o que garante uma abordagem completa de tudo o que está contido no diretório de origem.

```

remover_extras() {
    local backup_dir="$1"
    local trabalho_dir="$2"

    for item in "$backup_dir"/*; do
        local nome_base
        nome_base=$(basename "$item")

        #if [[ -f "$item" && ! -f "$trabalho_dir/$nome_base" ]] || esta_na_lista "$nome_base"; then
        if [[ ! -f "$item" ]]; then
            if [[ ! -f "$trabalho_dir/$nome_base" ]] || esta_na_lista "$nome_base"; then
                #remover ficheiro
                if [[ "$MODO_VERIFICAR" == false ]]; then
                    rm "$item"
                else
                    printf "rm %s\n" "$item"
                fi
            fi
        fi
    done

    #elif [[ -d "$item" && ! -d "$trabalho_dir/$nome_base" ]] || esta_na_lista "$nome_base"; then
    elif [[ -d "$item" ]]; then
        if [[ ! -d "$trabalho_dir/$nome_base" ]] || esta_na_lista "$nome_base"; then
            #remover pasta
            if [[ "$MODO_VERIFICAR" == false ]]; then
                rm -rf "$item"
            else
                printf "rm -rf %s\n" "$item"
            fi
        else
            #subd
            remover_extras "$item" "$trabalho_dir/$nome_base"
        fi
    fi
done
}

```

Figure 9/ Função remover_extras()

3) backup_summary.sh

Nesta última versão do programa, apenas são alteradas as funções de cópia e eliminação de ficheiros e diretórios, o que permite imprimir um resumo após a execução com informação sobre as operações efetuadas, incluindo contadores de erros, avisos, itens copiados, atualizados e apagados, bem como o tamanho total dos dados envolvidos em cada operação.

- Definição de Variáveis para Contadores

Foram definidos contadores de operações (ERROR, WARNINGS, COPIADOS, ATUALIZADOS, APAGADOS) e variáveis para registrar o tamanho total dos arquivos.

afetados por cada operação, como TAM_COP (arquivos copiados), TAM_ATU (arquivos atualizados) e TAM_APA (arquivos apagados)

- Cópia y remoção de arquivos

Para copiar os arquivos neste código, foi utilizado o mesmo código desenvolvido na versão anterior do programa, mas alterando algumas seções para que a função seja capaz de alterar os dados das variáveis que serão exibidas ao usuário que representam as operações que foram realizado no momento do backup.

A função utiliza um for para percorrer cada item (arquivo ou diretório) contido no diretório especificado por fonte.

Assim como no desenvolvimento anterior desta função, é aplicado um filtro para descartar arquivos contidos na BLACKLIST (função esta_na_lista) e trabalhar apenas com arquivos que cumpram o REGEX fornecido

Se o item é um arquivo (-f), verifica se o arquivo já existe no destino, caso não exista ou o arquivo na origem seja mais recente (usando -nt), o arquivo será copiado.

Se o item é um diretório (-d), a função chama a si mesma recursivamente para copiar o conteúdo do subdiretório, mantendo a estrutura hierárquica.

Atualiza contadores e exibe mensagens detalhadas, ajudando no monitoramento das ações realizadas (ou simuladas).

Contadores são atualizados e exibe mensagens detalhadas, ajudando no monitoramento das ações realizadas (ou simuladas).

```
#copyign
if [[ -f "$item" ]]; then
  if [[ ! -f "$destino/$nome_base" ]] || "$item" -nt "$destino/$nome_base" ]]; then
    if [[ "$item" -nt "$destino/$nome_base" && -f "$destino/$nome_base" ]]; then
      atualizacao=1
    fi
    if [[ "$MODO_VERIFICAR" == true ]]; then
      if [ $atualizacao == 0 ]; then
        printf "cp -a %s %s\n" "$item" "$destino/$nome_base"
        local_tam_cop=$((local_tam_cop + $(stat -c%s "$item")))
        ((local_copiados++))
      elif [ $atualizacao == 1 ]; then
        printf "cp -a %s %s\n" "$item" "$destino/$nome_base"
        local_tam_atu=$((local_tam_atu + $(stat -c%s "$item")))
        ((local_atualizados++))
      fi
    else
      mkdir -p "$destino"
      if cp -a "$item" "$destino/$nome_base"; then
        printf "cp -a %s %s\n" "$item" "$destino/$nome_base"
        if [ $atualizacao == 0 ]; then
          ((local_copiados++))
          local_tam_cop=$((local_tam_cop + $(stat -c%s "$item")))
        elif [ $atualizacao == 1 ]; then
          ((local_atualizados++))
          local_tam_atu=$((local_tam_atu + $(stat -c%s "$item")))
        fi
      else
        printf "Erro ao copiar '%s'\n" "$item"
        ((local_error++))
      fi
    fi
  elif [[ "$item" -ot "$destino/$nome_base" ]]; then
    printf "WARNING: backup entry %s is newer than %s; Should not happen\n" "$2" "$1"
    ((local_warnings++))
  fi
elif [[ -d "$item" ]]; then
  #subd
  copia_recursiva "$item" "$destino/$nome_base"
fi
```

Figure 10/ Modificações Função copia_recursiva

Para remover arquivos e diretórios que não estão mais presentes na pasta de trabalho, é aplicada a função `remove_extras` desenvolvida anteriormente com algumas alterações incorporadas.

A função usa um loop `for` para iterar sobre cada item (arquivo ou diretório) em `backup_dir`.

Se o item é um arquivo (`-f`), e no backup não existe no diretório de trabalho ou está listado para exclusão, ele será removido.

Se o item é um diretório (`-d`), o diretório no backup não tem correspondente no diretório de trabalho ou está na lista de exclusão, ele é removido recursivamente.

Finalmente, se houver arquivos removidos a função contabiliza o tamanho dos itens removidos (`tam_item`) para somá-los a `TAM_APA` e atualiza o contador de itens apagados (`APAGADOS`), para mostrá-los no final da execução do script

```
#if [[ -f "$item" && ! -f "$trabalho_dir/$nome_base" ]] || esta_na_lista "$nome_base"
if [[ -f "$item" ]]; then
    if [[ ! -f "$trabalho_dir/$nome_base" ]] || esta_na_lista "$nome_base"; then
        local tam_item #nao e redundante, e necessario guardar antes de apagar
        tam_item=$(stat -c%s "$item")
        #remover ficheiro
        if [[ "$MODULO_VERIFICAR" == false ]]; then
            rm "$item"
        else
            printf "rm '%s'\n" "$item"
        fi
        ((APAGADOS++))
        TAM_APA=$((TAM_APA + tam_item))
    fi
fi

#elif [[ -d "$item" && ! -d "$trabalho_dir/$nome_base" ]] || esta_na_lista "$nome_base"
elif [[ -d "$item" ]]; then
    if [[ ! -d "$trabalho_dir/$nome_base" ]] || esta_na_lista "$nome_base"; then
        #remover pasta
        if [[ "$MODULO_VERIFICAR" == false ]]; then
            rm -rf "$item"
        else
            printf "rm -rf '%s'\n" "$item"
        fi
        ((APAGADOS++))
        TAM_APA=$((TAM_APA + tam_item))
    else
        #subd
        remove_extras "$item" "$trabalho_dir/$nome_base"
    fi
fi
done
```

Figure 11/ Modificações Função `remove_extras()`

Validação da solução

Em cada uma das diferentes versões foram realizados testes e avaliação dos erros que possam ser gerados para garantir o correto funcionamento dos scripts.

1) backup_files.sh

- Testes

Teste Básico de Backup:

Criar um diretório de trabalho com alguns arquivos e um diretório de backup vazio. Executar o script e verificar se todos os arquivos foram copiados corretamente.

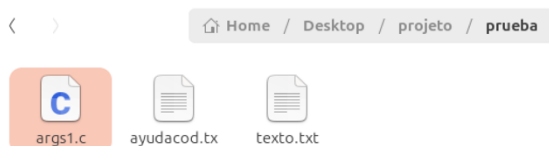


Figure 12/ Diretório de origem

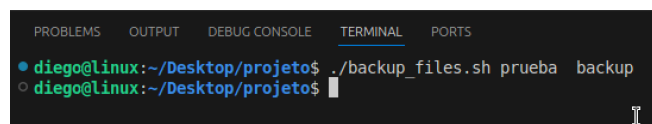


Figure 13/ Execute backup

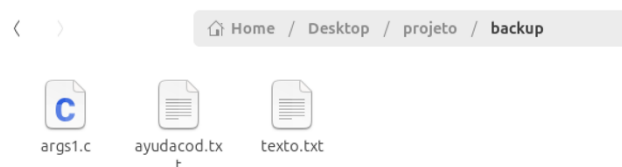


Figure 14/ Copia arquivos

Modo de Verificação (-c):

Executar o script com a opção -c e verificar se ele apenas exibe os comandos de cópia e remoção sem alterar o sistema de arquivos.

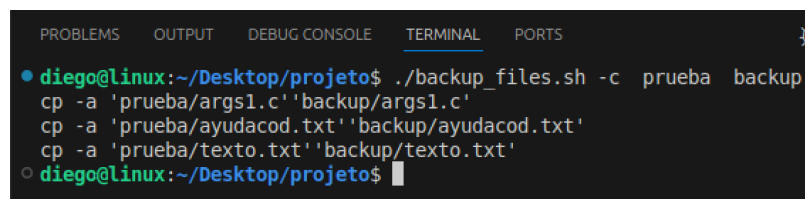


Figure 15/ Execute com -c

Teste de Remoção de Arquivos Obsoletos:

Adicionar um arquivo ao diretório de backup que não existe mais no diretório de trabalho. Executar o script e verificar se o arquivo é removido do backup.

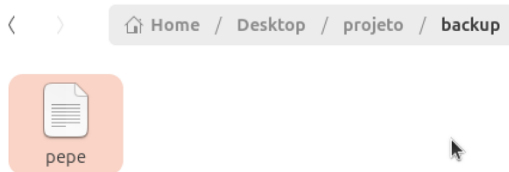


Figure 16/ Arquivo não incluído no diretório de origem

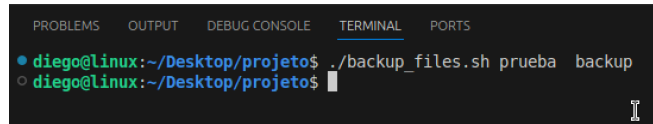


Figure 17/ Execute backup

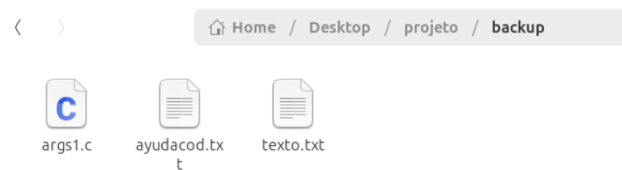


Figure 18/ Backup eliminando arquivo

- Erros

Como referido nos pontos anteriores, o código contém linhas para verificar possíveis erros e notificá-los, evitando assim problemas na execução do código. As imagens anexas mostram o que está descrito.

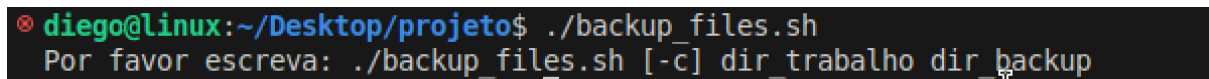


Figure 19/ Erro por não ter argumentos necessários

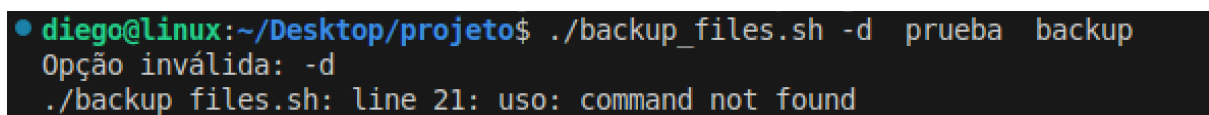


Figure 20/ Erro por ter argumento diferente a -c

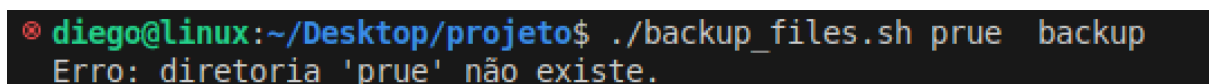


Figure 21/ Erro se diretoria não existisse

```

diego@linux:~/Desktop/projeto$ ./backup_files.sh -d prueba
Opção inválida: -d
./backup_files.sh: line 21: uso: command not found
cp: cannot create regular file '/args1.c': Permission denied
Erro ao copiar 'prueba/args1.c' para '/args1.c'
cp: cannot create regular file '/ayudacod.txt': Permission denied
Erro ao copiar 'prueba/ayudacod.txt' para '/ayudacod.txt'
cp: cannot create regular file '/texto.txt': Permission denied
Erro ao copiar 'prueba/texto.txt' para '/texto.txt'

```

Figure 22/ Erro na copia dos arquivos

2) backup.sh

- Testes

Teste de Cópia Inicial:

Objetivo: Verificar se o script copia todos os arquivos e diretórios do diretório de trabalho para o diretório de backup.

Procedimento: Crie alguns arquivos no diretório de trabalho e execute o script para fazer o backup inicial.

Resultado Esperado: Todos os arquivos e diretórios do diretório de trabalho devem ser replicados no diretório de backup.

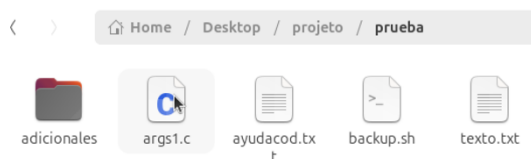


Figure 23/ Diretório de origem

```

diego@linux:~/Desktop/projeto$ ./backup.sh prueba backup
diego@linux:~/Desktop/projeto$

```

Figure 24/ Execute backup

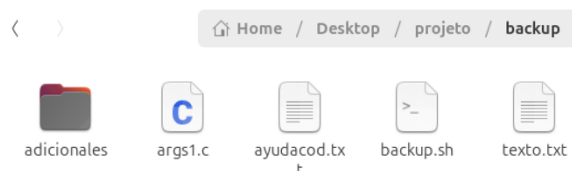


Figure 25/ Copia arquivos y subdirectorios

Teste com Regex para Seleção de Arquivos:

Objetivo: Verificar se a opção `-r` filtra corretamente os arquivos a serem copiados.

Procedimento: No diretório de trabalho, crie arquivos com extensões variadas (e.g., `.txt`, `.log`, `.jpg`). Execute o script com `-r ".*\.txt"` para copiar apenas arquivos `.txt`.

Resultado Esperado: Somente os arquivos `.txt` devem ser copiados para o diretório de backup.

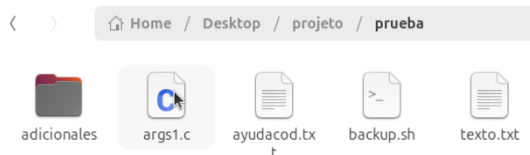


Figure 26/ Diretório de origem

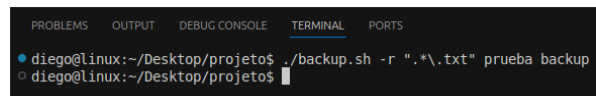


Figure 27/ Execute backup com filtro `-r`

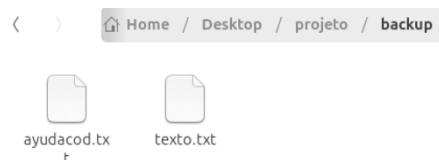


Figure 28/ Copia arquivos com filtro `-r`

Teste com Arquivo Blacklist:

Objetivo: Verificar se os arquivos listados no arquivo de blacklist são corretamente ignorados pelo script.

Procedimento: Crie um arquivo de blacklist que contenha alguns dos arquivos ou diretórios presentes no diretório de trabalho. Execute o script com a opção `-b blacklist.txt`.

Resultado Esperado: Os arquivos e diretórios listados na blacklist não devem ser copiados para o diretório de backup.

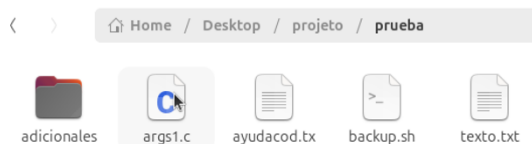


Figure 29/ Diretório de origem

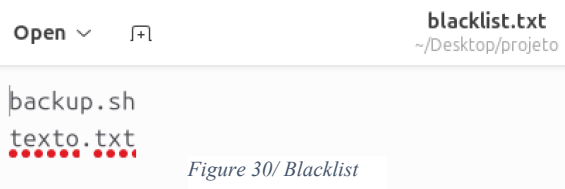
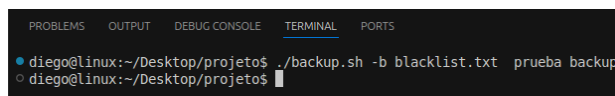


Figure 30/ Blacklist



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
diego@linux:~/Desktop/projeto$ ./backup.sh -b blacklist.txt prueba backup
diego@linux:~/Desktop/projeto$
```

Figure 31/ Execute backup com blacklist

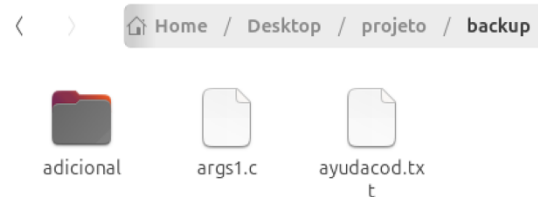
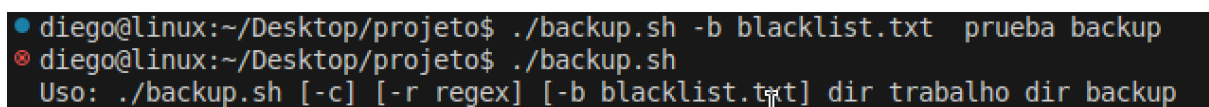


Figure 32/ Copia arquivos

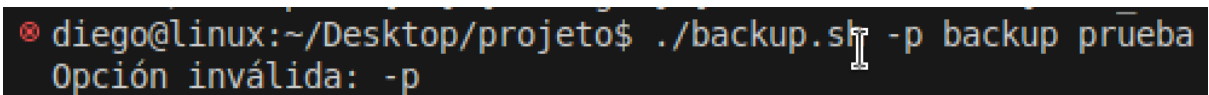
- Erros

Tratamento de erros apresentados na versão anterior, eles também estão presentes nesta versão do código



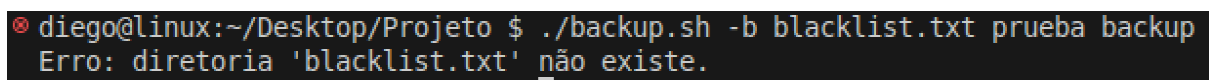
```
diego@linux:~/Desktop/projeto$ ./backup.sh -b blacklist.txt prueba backup
diego@linux:~/Desktop/projeto$ ./backup.sh
Uso: ./backup.sh [-c] [-r regex] [-b blacklist.txt] dir_trabalho dir_backup
```

Figure 33/ Erro por não ter argumentos necessários



```
diego@linux:~/Desktop/projeto$ ./backup.sh -p backup prueba
Opción inválida: -p
```

Figure 34/ Erro por ter argumento diferente a -c,-r ou -b



```
diego@linux:~/Desktop/Projeto $ ./backup.sh -b blacklist.txt prueba backup
Erro: diretoria 'blacklist.txt' não existe.
```

Figure 35/ Erro se blacklist não existe

3) Backup_summary.sh

Como grande parte das funções já foram verificadas em versões anteriores do código, para a versão final foram realizados apenas dois testes para verificar o correto funcionamento das novas implementações.

- Testes

Teste com Diretórios Vazios:

Objetivo: Confirmar que o script copia e remove diretórios vazios corretamente, e as operações executadas são apresentadas

Procedimento: Crie diretórios vazios na pasta de trabalho, execute o script, depois remova esses diretórios e execute o script novamente.

Resultado Esperado: Diretórios vazios devem ser copiados e, após a exclusão, removidos do backup, além disso, as operações executadas devem ser impressas no terminal.

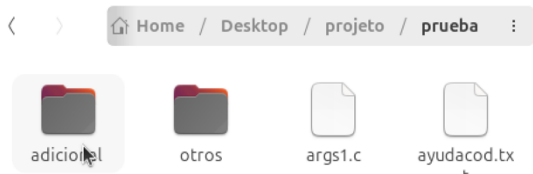


Figure 36/ Diretório de origem

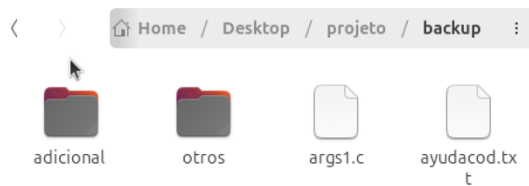


Figure 37/ Cópia arquivos

```
diego@linux:~/Desktop/projeto$ ./backup_summary.sh prueba backup
While backuping prueba: 0 Errors; 0 Warnings; 0 Updated (0B); 3 Copied (0B); 0 Deleted (0B)
```

Figure 38/ Execute do backup e informações de operações feitas

Teste com Arquivos Grandes:

Objetivo: Verificar a capacidade do script de lidar com arquivos grandes.

Procedimento: Coloque um arquivo grande na pasta de trabalho e execute o script.

Resultado Esperado: O script deve processar o arquivo grande corretamente, e o contador de tamanho (TAM_COP ou TAM_ATU) deve refletir o tamanho do arquivo.

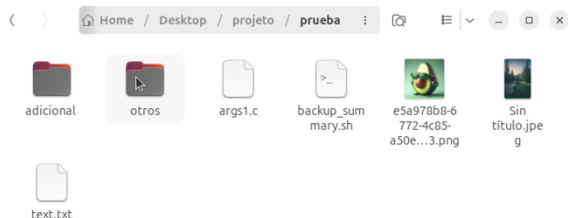


Figure 39/ Diretório de origem

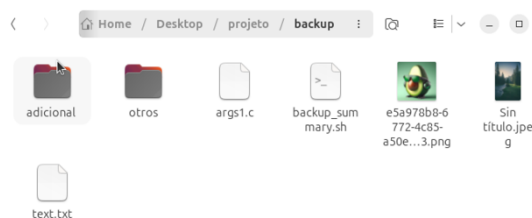


Figure 40/ Cópia y arquivos

```
diego@linux:~/Desktop/projeto$ ./backup_summary.sh prueba backup
While backuping prueba: 0 Errors; 0 Warnings; 0 Updated (0B); 6 Copied (633746B); 0 Deleted (0B)
```

Figure 41/ Execute do backup e informações de operações feitas

Erros

Nesta versão final do código, apenas um novo erro pode ser gerado além dos demais já abordados nas versões anteriores do código.

```
diego@linux:~/Desktop/projeto$ ./backup_summary.sh prueba backup
WARNING: backup entry 'backup/args1.c' is newer than 'prueba/args1.c'; should not happen
While backuping prueba: 0 Errors; 1 Warnings; 0 Updated (0B); 0 Copied (0B); 0 Deleted (0B)
```

Figure 42/ Erro se backup contiver arquivos mais atuais que o origem

4) backup_check.sh

Como complemento e para ajudar a validar o backup realizado pelos programas, foi desenvolvido este script, o qual verifica se os arquivos em dois diretórios são diferentes comparando seus hashes MD5 (possível com comando md5sum). Ele usa uma abordagem iterativa para cada arquivo no diretório de trabalho e compara com seu correspondente no diretório de backup.

Através da função `verifica_diferenca` que recebe dois parâmetros:

- **arquivo_trabalho:** Caminho do arquivo no diretório de trabalho.
- **arquivo_backup:** Caminho do arquivo correspondente no diretório de backup.

Calcula o hash MD5 dos dois arquivos usando `md5sum` e compara os hashes.

Se forem diferentes, imprime uma mensagem indicando que os arquivos divergem, caso contrário, imprime que são iguais.

```
#!/bin/bash

#verificacoes
if [[ "$#" -lt 2 ]]; then #numero argumentos
    printf "Por favor escreva: %s 1 2\n" "$0"
    exit 1
fi

#comparar hash dos arquivos
verifica_diferenca() {
    local arquivo_trabalho="$1"
    local arquivo_backup="$2"

    #calcular hash MD5
    local md5_trabalho
    local md5_backup
    md5_trabalho=$(md5sum "$arquivo_trabalho" | awk '{print $1}')
    md5_backup=$(md5sum "$arquivo_backup" | awk '{print $1}')

    #comparar
    if [[ "$md5_trabalho" != "$md5_backup" ]]; then
        printf "%s %s differ.\n" "$arquivo_trabalho" "$arquivo_backup"
    else
        printf "%s %s iguais\n" "$arquivo_trabalho" "$arquivo_backup"
    fi
}

#recorrer
find "$1" -type f | while read -r arquivo_trabalho; do
    #caminho correspondente
    arquivo_backup="${2}${arquivo_trabalho#1}"

    #comparar se existe
    if [[ -f "$arquivo_backup" ]]; then
        verifica_diferenca "$arquivo_trabalho" "$arquivo_backup"
    fi
done
```

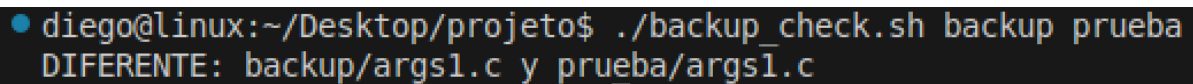
Figure 43/ Código backup_check

- Testes

Objetivo: verificar se os backups foram feitos corretamente ou se os arquivos são diferentes.

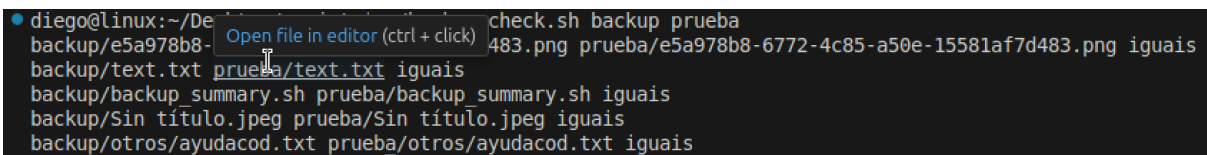
Procedimento: faça backup de um diretório e execute o script, altere alguns dos arquivos de backup e execute o script.

Resultado: Será impresso no terminal se os arquivos são iguais ou diferentes, demonstrando se o backup foi realizado corretamente ou não.



```
diego@linux:~/Desktop/projeto$ ./backup_check.sh backup prueba  
DIFERENTE: backup/args1.c y prueba/args1.c
```

Figure 44/ Arquivos diferentes



```
diego@linux:~/Desktop/projeto$ ./backup_check.sh backup prueba  
backup/e5a978b8-483.png prueba/e5a978b8-6772-4c85-a50e-15581af7d483.png iguais  
backup/text.txt prueba/text.txt iguais  
backup/backup_summary.sh prueba/backup_summary.sh iguais  
backup/Sin título.jpeg prueba/Sin título.jpeg iguais  
backup/otros/ayudacod.txt prueba/otros/ayudacod.txt iguais
```

Figure 45/ Arquivos iguais

Conclusão

Concluindo, a tarefa proposta foi cumprida com grande êxito. Além de alcançar um código plenamente funcional, o processo de desenvolvimento dos scripts proporcionou uma oportunidade valiosa para aprofundar os conhecimentos em programação Bash, contribuindo significativamente a capacidade de solucionar problemas de forma eficiente e o trabalho em grupo.

Bibliografia

- <https://www.geeksforgeeks.org/bash-scripting-introduction-to-bash-and-bash-scripting/>
- <https://stackoverflow.com/>
- <https://kodekloud.com/blog/regex-shell-script/>
- <https://lernentec.com/post/2023/08/essential-bash-flags/>