## Exercícios

Os exercícios abaixo devem ser solucionados com línguagens Shell, como `bash`, `nushell`, `zsh`, `fish`, etc.

Na décima página, um exercício é proposto – *e erronamente solucionado* – com o intuito de entender melhor os filtros em Shell scripting:

0. Suponha que exista um arquivo chamado `hotel.txt` com 100 linhas de dados. Imprima começando da linha 20 à linha 30 e armazene este resultado em um novo arquivo `hlist`.

```bash
$ tail +20 < hotel.txt | head -10 > hlist
```
BASH

```nushell
$ open hotel.txt | lines | skip 20 | first 10 | save hlist
```
NUSHELL

---

Os exercícios a seguir são da página 44 em diante, e devem ser feitos escrevendo um script shell.

1. Adicione dois números que são recebidos pela linha de comando como argumentos, e se esses dois números não são dados, mostre um erro e seu uso correto:

```nushell
#!/usr/bin/env nu

# Add two numbers and return its sum.
def main [
  a: int # First number to add.
  b: int # Second number to add.
]: nothing -> int {
  $a + $b
}
```
NUSHELL

2. Retorne o maior número entre três argumentos dados pela linha de comando, e se três argumentos não forem dados, mostre um erro e seu uso correto.

```nushell
#!/usr/bin/env nu

# Return the greastest number between the three given.
def main [
  a: int # First number to analyse.
  b: int # Second number to analyse.
  c: int # Third number to analyse.
]: nothing -> int {
  [ $a $b $c ] | sort | last
}
```
NUSHELL

3. Imprima a sequência 5, 4, 3, 2, 1 utilizando a repetição `while`:

```bash
#!/usr/bin/env bash

m=5;
```
BASH

```
while [ $m -gt 0 ]
do
  echo "$m"
  m=`expr $m - 1`
done
```

```
#!/usr/bin/env nu

# Create the sequence 5, 4, 3, 2, 1 using the while loop.
def main []: nothing -> list<int> {
  mut result = []

  mut i = 5
  while $i != 0 {
    $result = $result ++ [ $i ]
    $i = $i - 1
  }

  $result
}
```
NUSHELL

3.1. Há outras formas de se resolver sem `while`, quais seriam?

```
#!/usr/bin/env bash

while ((m > 0)); do
  echo "$m"
  m=`expr $m - 1`
done
```
BASH

```
#!/usr/bin/env nu

# Create the sequence 5, 4, 3, 2, 1 using the command seq.
def main []: nothing -> list<int> {
  seq 5 -1 1
}
```
NUSHELL

3.2. E para um máximo e mínimo qualquer dado pela linha de comando, como resolver?

```
#!/usr/bin/env bash

if [ $# -lt 2 ]
then
  echo "MAX or MIN missing"
  exit 1
fi

i=$1
```
BASH

```
while [ $i -ge $2 ]
do
  echo "$i"
  i=`expr $i - 1`
done
```

```
#!/usr/bin/env nu

# Create a decreasing sequence starting from `max` and ending in `min`.
def main [
  max: int # Start of the sequence.
  min: int # End of the sequence.
]: nothing -> list<int> {
  seq $max -1 $min
}
```

NUSHELL

3.3. Imprimir a ordem crescente quando o primeiro argumento for menor que o segundo, senão imprimir a ordem decrescente.

```
#!/usr/bin/env bash

if [ $# -lt 2 ]
then
  echo "MAX or MIN missing"
  exit 1
fi

if [ $2 -gt $1 ]
then
  i=$1
  j=$2

  while [ $i -le $j ]
  do
    echo "$i"
    i=`expr $i + 1`
  done

elif [ $1 = $2 ]
then
  echo "$1"
else
  i=$1
  j=$2

  while [ $i -ge $j ]
  do
    echo "$i"
    i=`expr $i - 1`
  done
fi
```

BASH

4. Usando a palavra-chave `case` performe operações matemáticas básicas como adição (+), subtração (-), multiplicação (x), e divisão (/).

```nushell
#!/usr/bin/env nu

# Using case (match) keyword to perform basic math operations such as addition
(+), subtraction (-), multiplication (x), and division (/).
def main [
  a: int # First operand.
  operator: string # Operator.
  b: int # Second operand.
]: nothing -> int {
  match $operator {
    '+' => { $a + $b },
    '-' => { $a - $b },
    'x' => { $a * $b },
    '/' => { $a / $b },
    _ => { NaN }
  }
}
```

4.1. Ao invés de apenas três argumentos, resolva para um número ilimitado de argumentos (podendo ter $1, 2, \cdots$ argumentos).

5. Mostre a data, tempo, usuário, e diretório atual.

```nushell
#!/usr/bin/env nu

# Show the current date, time, username, and current directory.
def main []: nothing -> any {
  {
    date: (date now | into string)
    user: (whoami)
    directory: $env.PWD
  }
}
```

6. Faça o reverso do primeiro argumento dado.

```nushell
#!/usr/bin/env nu

# Reverse the given argument.
export def main [
  arg: string # String to be reversed.
]: nothing -> any {
  $arg | str reverse
}
```

7. Dado um número como argumento, calcule a soma de todos os digitos.

```nushell
#!/usr/bin/env nu
```

```
# Given number, calculate the sum of its digits.
export def main [
  arg: int # Number to sum each digit.
]: nothing -> any {
  $arg | into string | split chars | into int | math sum
}
```

8. Performe aritmética real (números com pontos decimais) dado os argumentos.

**Mesma solução da questão quatro para Nushell.**

9. Calcule a seguinte operação entre os dois números reais: $5.12 + 2.5$.

```
$ 5.12 + 2.5
```
NUSHELL

```
$ echo 5.12 + 2.5 | bc
```
BASH

10. Como guardar o resultado de uma operação no conjunto dos números reais:

```
$ let a = 5.66
$ let b = 8.67
$ let c = $a + $b
```
NUSHELL

11. Determine se um arquivo existe dado seu caminho pela linha de comando.

```
#!/usr/bin/env nu

# Given path of a possible file, verify if it exists.
export def main [
  path: string # Path to verify if file exists.
]: nothing -> bool {
  ($path | path type) == "file"
}
```
NUSHELL

12. Dado um argumento, verifique se ele contém o caractére asterisco (*), se não conter, adicione ao fim e o imprima. Se conter, imprima: “Symbol is not required.”.

```
#!/usr/bin/env nu

# Add an asterisk to the argument.
export def main [
  arg: string # String to verify.
]: nothing -> string {
  if ($arg | str contains '*') {
    "Symbol is not required."
  } else {
    $arg + '*'
```
NUSHELL

```
  }
}
```

13. Dado o diretório de um arquivo como o primeiro argumento, imprima seu conteúdo a partir do segundo argumento (o ínicio), até o terceiro argumento, a quantidade de linhas seguintes.

```nushell
#!/usr/bin/env nu

# Add an asterisk to the argument.
export def main [
  path: string # String to verify.
  start: int # Start of the file.
  end: int # End of the file.
]: nothing -> string {
  if ($path | path type) != "file" {
    error make {
      msg: $'Not a file.'
      label: {
        text: $'($path) is not a valid value.'
        span: (metadata $path).span
      }
    }
  }

  open $path | lines | skip $start | first $end
}
```

14. Implemente um programa que aceite as seguintes flags:

- `-c` para limpar a tela (de *clear*);
- `-d` mostrar a lista de arquivos no diretório atual;
- `-m` comecar o programa `mc` (Midnight Commander Shell) se estiver instalado;
- `-e` comecar o editor padrão do sistema.

```nushell
#!/usr/bin/env nu

# Quick launcher of certain commands.
export def main [
  --clear (-c) # Clear the screen.
  --list-directory (-d) # List the files of the current directory.
  --midnight-commander (-m) # Execute the Midnight Commander Shell.
  --editor (-e) # Execute the default editor of the system.
]: nothing -> any {
  if $clear {
    clear
  } else if $list_directory {
    ls ./
  } else if $midnight_commander {
    exec mc
  } else if $editor {
    exec $env.EDITOR
```

```
  }
}
```

15. Escreva um programa nomeado `sayHello`, e coloque-o para executar assim que o shell iniciar. Então imprima qualquer uma das seguintes mensagens no seu sistema (da forma que você quiser) de acordo com o horário:
- “Bom dia!”,
- “Boa tarde!”,
- “Boa noite!”.

16. Imprima a mensagem “Bom Dia, Mundo!”, em negrito e efeito piscante, com cores diferentes como vermelho, marron, etc, utilizando o comando `echo`.

17. Implemente um programa que rode de fundo (como serviço) que imprimirá continuamente o tempo atual no lado direito acima da tela, enquanto o(a) usuário(a) pode usar normalmente o sistema.

18. Escreva um programa que implemente utilitários de diálogo: items de menu e ações de acordo com o que foi selecionado como o seguinte:

| **Item Menu** | **Propósito** | **Ações para o Item Menu** |
|---|---|---|
| Data/tempo | Ver o tempo e data atual. | Data e tempo deve ser mostrado usando uma caixa de informação numa utilidade de diálogo. |
| Calendário | Ver o calendário atual | O calendário deve ser mostrado usando uma caixa de informação numa utilidade de diálogo. |
| Delete | Deletar o arquivo selecionado. | Primeiro peça ao usuário o nome do diretório onde todos os arquivos estão presentes, se não tiver nome, então assume-se o diretório atual. Então mostre todos osarquivos usando apenas aquele diretório, os arquivos devem ser selecionáveis. Após o usuário escolher um, confirme com o usuário e então se confiramdo, delete o arquivo, reporte erros se não conseguir. |
| Exit | Sair deste programa | Sair do programa atual (do script que estiver rodando) |

19. Escreva um programa que mostre as informações do sistema, como:
- Usuário logado atual e seu nome no sistema;
- Shell atual;
- Diretório *home*;
- Tipo do Sistema Operacional;
- Configurações de caminho;
- Diretório atual;

- Número total de usuários de logados;
- Informações do Linux, como versão, número de lançamento, etc;
- Todos os *shells* disponíveis.
- Configurações do mouse;
- Informações do processador como tipo, velocidade, etc;
- Informações de memória;
- Informações do dispositivo físico de armazenamento, tamanho total, memória cache, modelo, etc;
- Sistema de arquivos montado.