Capítulo 3: Introdução a Script

Type	Leitura
Materials	Capítulo 3.pdf
Reviewed	☐

▼ ***Exemplos - Práticos***

Exemplo 2: Comparar dois números

Script:

```
NUM1=10
NUM2=5

if test $(expr $NUM1 - $NUM2) -gt 0; then
    echo "NUM1 é maior que NUM2."
else
    echo "NUM1 não é maior que NUM2."
fi



# Vê se voce passou parametro ou não
#
#if test "$#" -eq 0
```

```
#!/bin/bash

AMOTRA=$1
```

```
if expr 1 + ${AMOSTRA}
then
    echo "${AMOTRA} é um número."
else
    echo "${AMOSTRA} não é um numero"
fi
```

BASH

- Conjunto de comandos que você escreve em um arquivo para automatizar tarefas repetitivas ou complexas no computador
- Todo script tem a extensão `.sh`
- Para executar, basta fazer: `./meu_script.sh`
- Antes de executar o script, dê um `chmod +x meu_script.sh`

Formas de criar variável

```
#!/bin/bash



# Isso é um comentário



# OBS: VOCE PODE PEGAR O RESULTADO USANDO OU PARÊNTESE OU ASP
listagem=$(ls -l /usr)
data_atual='date'
```

```
total_arquivos=$(ls | wc -l)

echo "Existe $total_arquivos arquivos neste diretório."









echo "A listagem é:"
echo $listagem

# Mostra a data atual
echo "A data atual é: $date"
```

Variável Local

- Uma variável local é acessível apenas dentro do bloco ou função onde foi declarada

```
#!/bin/bash

# Função que define uma variável local
minha_funcao() {
    local variavel_local="Eu sou uma variável local"
    echo $variavel_local
}

# Chama a função
minha_funcao
```

```
# Tenta acessar a variável local fora da função
echo $variavel_local  # Não vai imprimir nada, porque variave
```

Variável Global

- Uma variável global é acessível em qualquer parte do script após sua declaração

```
#!/bin/bash

# Define uma variável global
variavel_global="Eu sou uma variável global"

# Função que acessa a variável global
minha_funcao() {
    echo $variavel_global
}

# Chama a função
minha_funcao

# Acessa a variável global fora da função
echo $variavel_global
```

Read

- ler a entrada do usuário ou de um arquivo. Ele permite que você capture a entrada e a armazene em variáveis para uso posterior no script.

```
read [opções] variável
```

```
#!/bin/bash

# Usa a opção -p para exibir um prompt antes de ler a entrada
read -p "Digite sua idade: " idade

# Imprime a idade do usuário
echo "Você tem $idade anos."
```

Opções Comuns:

`-p "prompt"`: Exibe um prompt antes de ler a entrada.

```
#!/bin/bash

#Exibe um prompt e aguarda a entrada do usuário

read -p "Digite o seu nome: " nome

#Exibe o nome digitado

echo "Olá, $nome!"
```

`-s`: Oculta a entrada (útil para senhas).

```
#!/bin/bash

# Exibe um prompt e oculta a entrada do usuário
read -s -p "Digite a sua senha: " senha

# Adiciona uma nova linha para não colar a saída no prompt
echo -e "\nSenha capturada."
```

`-r:` Não trata os caracteres de barra invertida (\) como especiais.

```
#!/bin/bash

# Solicita a entrada do usuário e não interpreta barras inver
read -r -p "Digite uma string com barras invertidas (ex: C:\\

# Exibe a entrada capturada
echo "Você digitou: $entrada"
```

`-n número`: Lê um número específico de caracteres.

```
#!/bin/bash

# Solicita que o usuário insira três caracteres
echo "Digite os primeiros 3 caracteres da sua senha:"

# Lê exatamente 3 caracteres da entrada
read -n 3 senha

# Exibe os caracteres lidos
echo -e "\nOs caracteres que você digitou foram: $senha"
```

`-t`: Define um tempo limite (em segundos) para a entrada.

```
#!/bin/bash

# Exibe um prompt e aguarda 5 segundos para a entrada do usuá
read -t 5 -p "Digite algo em 5 segundos: " resposta

# Verifica se a entrada foi feita a tempo
if [ -z "$resposta" ]; then
```

```
    echo "Tempo esgotado!"
else
    echo "Você digitou: $resposta"
fi
```

`-d`: Define um delimitador de entrada diferente de Enter.

```
#!/bin/bash

# Solicita a entrada do usuário com vírgula como delimitador
read -d "," -p "Digite algo e finalize com uma vírgula: " ent

# Exibe a entrada capturada
echo "Você digitou: $entrada"
```

Parâmetros em Scripts de Shell

- shell podem receber dados diretamente via linha de comando

Principais Parâmetros:

- `$0`: O nome do script.
- `$1, $2, ..., $N`: Os parâmetros passados na linha de comando, onde `$1` é o primeiro parâmetro, `$2` é o segundo, e assim por diante.
- `$@` ou `$*`: Todos os parâmetros passados a partir de `$1`.
- `$#`: O número de parâmetros passados.
- `$$`: O PID (Process ID) do processo atual.

```
#!/bin/bash

echo "Nome do script: $0"
echo "Primeiro parâmetro: $1"
echo "Segundo parâmetro: $2"
```

```
echo "Todos os parâmetros: $@"
echo "Número de parâmetros: $#"
echo "PID do processo atual: $$"

#Para executar este script, você pode passar alguns parâmetro
./meu_script.sh param1 param2 param3
./meu_script.sh 5 6 7 "teste" semaspas



Nome do script: ./meu_script.sh
Primeiro parâmetro: param1
Segundo parâmetro: param2
Todos os parâmetros: param1 param2 param3
Número de parâmetros: 3
PID do processo atual: 12345  # (O valor do PID será diferent
```

Estrutura Condicional e Repetição

```
if [ condição ]; then #ENTÃO
    comandos
elif [ outra_condição ]; then #ENTÃO
    outros_comandos
else
    comandos_alternativos
fi




EXEMPLO 1:



#!/bin/bash
```

```
num=10

if [ $num -gt 5 ]; then
    echo "O número é maior que 5"
elif [ $num -eq 5 ]; then
    echo "O número é igual a 5"
else
    echo "O número é menor que 5"
fi
```

EXEMPLO 2:

```
#!/bin/bash

letra="a"

case $letra in
    "a")
        echo "A letra é a"
        ;;
    "b")
        echo "A letra é b"
        ;;
    *)
        echo "A letra não é nem a nem b"
        ;;
esac
```

-eq	Igual
-ne	Diferente
-gt	Maior
-lt	Menor
-o	Ou
-d	Se for um diretório
-e	Se existir
-z	Se estiver vazio
-f	Se conter texto
-o	Se o usuário for o dono
-r	Se o arquivo pode ser lido
-w	Se o arquivo pode ser alterado
-x	Se o arquivo pode ser executado

```
  GNU nano 7.2
#!/bin/bash

# Solicita ao usuário que digite um número
echo "Digite um número:"
read num



# Verifica se o número é igual a 10
if [ "$num" -eq 10 ]; then
    echo "O número é igual a 10."
fi
```

```
# Verifica se o número não é igual a 10
if [ "$num" -ne 10 ]; then
    echo "O número não é igual a 10."
fi



# Verifica se o número é maior que 10
if [ "$num" -gt 10 ]; then
    echo "O número é maior que 10."
fi



# Verifica se o número é menor que 10
if [ "$num" -lt 10 ]; then
    echo "O número é menor que 10."
fi



# Verifica se o número é zero
if [ -z "$num" ]; then
    echo "Nenhum número foi digitado."
fi



# Verifica se o número é maior que 10 OU menor que 5
```

```
if [ "$num" -gt 10 ] || [ "$num" -lt 5 ]; then
    echo "O número é maior que 10 ou menor que 5."
fi




# Verifica se o diretório especificado existe
echo "Digite o nome de um diretório:"
read dir
if [ -d "$dir" ]; then
    echo "O diretório $dir existe."
else
    echo "O diretório $dir não existe."
fi







# Verifica se o arquivo especificado existe
echo "Digite o nome de um arquivo:"
read file
if [ -e "$file" ]; then
    echo "O arquivo $file existe."
else
    echo "O arquivo $file não existe."
fi




# Verifica se o arquivo especificado está vazio
if [ -s "$file" ]; then
    echo "O arquivo $file não está vazio."
else
```

```
    echo "O arquivo $file está vazio ou não existe."
fi




# Verifica se o arquivo especificado é um arquivo que contem
if [ -f "$file" ]; then
    echo "$file é um arquivo que tem texto."
else
    echo "$file não é um arquivo que nao tem texto."
fi




# Verifica se o arquivo especificado pode ser lido
if [ -r "$file" ]; then
    echo "$file pode ser lido."
else
    echo "$file não pode ser lido."
fi




# Verifica se o arquivo especificado é gravável
if [ -w "$file" ]; then
    echo "$file é gravável."
else
    echo "$file não é gravável."
fi
```

```
# Verifica se o arquivo especificado é executável
if [ -x "$file" ]; then
    echo "$file é executável."
else
    echo "$file não é executável."
fi
```

REPETIÇÃO

- While, For

```
#!/bin/bash

for i in {1..10}; do
    echo "Número: $i"
done
```

```
#!/bin/bash

contador=1

until [ $contador -gt 5 ]; do
    echo "Contador: $contador"
    contador=$((contador + 1))
done
```

```
#!/bin/bash

contador=1
```

```
while [ $contador -le 5 ]; do
    echo "Contador: $contador"
    contador=$((contador + 1))
done
```

CASE

Exemplo 1:

```
#!/bin/bash

echo "Digite um número entre 1 e 3:"
read numero

case $numero in
    1)
        echo "Você escolheu o número 1."
        ;;
    2)
        echo "Você escolheu o número 2."
        ;;
    3)
        echo "Você escolheu o número 3."
        ;;
    *)
        echo "Número inválido!"
        ;;
esac
```

Exemplo 2:

```bash
#!/bin/bash

# Simula ações de gerenciamento de serviço
echo "Digite a ação (start, stop, restart, status):"
read action

case $action in
    start)
        echo "Iniciando o serviço..."
        # Comando real poderia ser: systemctl start <serviço>
        ;;
    stop)
        echo "Parando o serviço..."
        # Comando real poderia ser: systemctl stop <serviço>
        ;;
    restart)
        echo "Reiniciando o serviço..."
        # Comando real poderia ser: systemctl restart <serviç
        ;;
    status)
        echo "Verificando o status do serviço..."
        # Comando real poderia ser: systemctl status <serviço
        ;;
    *)
        echo "Ação inválida! Use: start, stop, restart ou sta
        ;;
esac
```

- `read numero`: Lê a entrada do usuário.
- Cada padrão termina com `)` e os comandos são executados se o padrão corresponder.
- `*`: Corresponde a qualquer valor não especificado nos padrões anteriores (equivalente ao "default").
- `;;`: Finaliza os comandos de cada caso.

HORA

```
$ date +'Hoje é %A, %d de %B de %Y, o %j dia do ano, as %
H:%M'
```

TRATAMENTO DE ERROS:

```
#!/bin/bash

comando
if [ $? -ne 0 ]; then
    echo "Erro: O comando falhou."
fi
```

```
#!/bin/bash
set -e

comando_que_pode_falhar
echo "Este comando não será executado se o anterior falhar."
```

```
#!/bin/bash
set -o pipefail

comando1 | comando2 | comando3
if [ $? -ne 0 ]; then
    echo "Erro em um dos comandos no pipeline."
fi
```

```
#!/bin/bash

if [ ! -f "arquivo.txt" ]; then
```

```
echo "Arquivo não encontrado!"
exit 1
fi
```

OBS: `$?` é uma variável especial que armazena o código de saída do último comando executado. O código de saída é um número que indica se o comando foi bem-sucedido ou se ocorreu um erro.