

SISTEMAS OPERACIONAIS

AULA 12: SHELL SCRIPTS

2 de julho de 2025

Prof. Me. José Paulo Lima

IFPE Garanhuns



**INSTITUTO
FEDERAL**
Pernambuco

SUMÁRIO



Sobre o Shell

- Arquivos de configuração

Sobre os scripts

- Estrutura

- Variáveis

 - Operações

- Estrutura condicional - SE

 - Operadores

 - Estrutura

- Estrutura condicional - CASO

- Estrutura de repetição - PARA

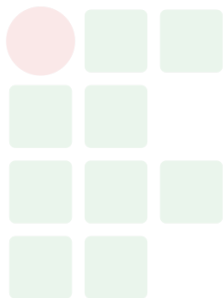
- Estrutura de repetição - ENQUANTO

- Parâmetros

- Funções

Referências

SOBRE O SHELL



**INSTITUTO
FEDERAL**
Pernambuco

SOBRE O SHELL



Tanenbaum e Bos (2016, p. 1) define:

“O programa com o qual os usuários interagem, normalmente chamado de **shell** (ou interpretador de comandos) quando ele é baseado em texto e de **GUI (Graphical User Interface)** quando ele usa ícones, na realidade não é parte do sistema operacional, embora use esse sistema para realizar o seu trabalho.”

- ▶ Implementa uma linguagem simples de programação que permite desenvolver pequenos programas, conhecidos como scripts;
- ▶ É uma interface usuário-sistema operacional, possui diversos comandos internos que permitem solicitar serviços do SO;
- ▶ O bash é um dos vários shells disponíveis;
 - ▶ Desenvolvido por Stephen Bourne (AT&T) em 1977, E significa **Bourne Again Shell**.

SOBRE O SHELL

ARQUIVOS DE CONFIGURAÇÃO



- ▶ A configuração do ambiente de trabalho de um usuário é feita por meio de três arquivos especiais, presentes no diretório /home do usuário:
 - ▶ `~/.bash_profile`
 - ▶ `~/.bashrc`
 - ▶ `~/.bash_logout`
- ▶ Esses arquivos permitem a execução de comandos em momentos distintos.

SOBRE O SHELL

ARQUIVOS DE CONFIGURAÇÃO



1. `~/.bash_profile`

- ▶ É processado toda vez que se efetua *login*;
 - ▶ Informações adicionadas a este arquivo não entrarão em vigor até que ele seja lido novamente, sendo necessário efetuar *logout* e *login* em seguida.
- ▶ Os comandos “.” e “source” podem ser usados para processar alterações do arquivo `~/.bash_profile`.

2. `~/.bashrc`

- ▶ É processado toda vez que um novo shell (subshell) é acionado.

3. `~/.bash_logout`

- ▶ É processado imediatamente antes de ser efetuado *logout*;
 - ▶ Assim, comandos para remover arquivos temporários poderão ser executados antes do encerramento da sessão.

SOBRE O SHELL

ARQUIVOS DE CONFIGURAÇÃO

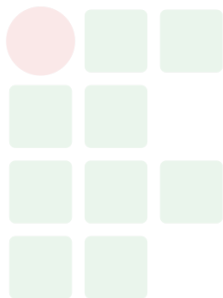


- Pode ser conveniente colocar no arquivo `~/ .bashrc` todas as informações de personalização e chamar, no arquivo `~/ .bash_profile`, o arquivo `~/ .bashrc`:

```
source ~/.bashrc
```

Para que assim, ao se iniciar um novo shell, o ambiente já esteja configurado.

SOBRE OS SCRIPTS



**INSTITUTO
FEDERAL**
Pernambuco

SOBRE OS SCRIPTS



Definição:

São arquivos simples contendo sequências de comandos, porém possuem estruturas de controle, como condicionais e laços de repetição.

- ▶ Forma fácil de automatizar tarefas:
 - ▶ Interage com os comandos do Linux;
 - ▶ Permite manipulação da saída dos comandos.
- ▶ Linguagem Interpretada.

SOBRE OS SCRIPTS

ESTRUTURA

► Início

- Indica o shell utilizado para interpretar o script:

```
#!/bin/bash
```

► Meio

- Comandos
 - Qualquer comando Linux.
- Estruturas de controle
 - Decisões;
 - Repetições.
- Comentários
 - Após o caractere "#";
 - Pode ser no início da linha ou após um comando;
 - Texto após "#" é ignorado pelo interpretador.

► Fim

- Não é obrigatório inserir:

```
exit 0
```

SOBRE OS SCRIPTS

DESENVOLVIMENTO DO PRIMEIRO SCRIPT



1. Criar um novo arquivo em um editor de texto qualquer:

```
nano meuPrimeiroScript.sh
```

2. Digitar os comandos do script, por exemplo:

```
#!/bin/bash  
echo Hello World  
exit 0
```

- ▶ Este script que imprime um texto na tela;
- ▶ Comando echo imprime na tela a frase passada como parâmetro levando o cursor para a próxima linha.

SOBRE OS SCRIPTS

DESENVOLVIMENTO DO PRIMEIRO SCRIPT

3. Executar o script:

- ▶ Necessita de permissão de execução:

```
chmod ugo+x meuPrimeiroScript.sh
```

- ▶ Há três formas de execução:

```
./meuPrimeiroScript.sh  
source meuPrimeiroScript.sh  
bash meuPrimeiroScript.sh
```

- ▶ Quando se utiliza “./meuPrimeiroScript.sh” ou “source meuPrimeiroScript.sh”, o shell script é executado no shell corrente;
- ▶ Na realidade a forma “./meuPrimeiroScript.sh” indica apenas que o comando a ser executado é local;
- ▶ Quando se utiliza a forma “bash meuPrimeiroScript.sh”, o shell script é executado em uma cópia do shell (subshell), com o controle sendo passado ao shell-pai após seu término.

SOBRE OS SCRIPTS

VARIÁVEIS



- ▶ Variáveis não tem tipo definido
 - ▶ São strings que armazenam dados;
 - ▶ Se forem somente dígitos (números), o bash permite operações inteiras (soma, subtração);
 - ▶ O valor pode ser uma frase, números e até outras variáveis ou comandos Linux.
- ▶ Operações:
 1. Atribuir valor a variável;
 2. Acessar valor da variável;
 3. Imprimir na tela;
 4. Ler uma variável do teclado;
 5. Executar comandos.

SOBRE OS SCRIPTS

OPERAÇÕES COM VARIÁVEIS

1. Atribuir valor a variável:

- ▶ A declaração deve ocorrer sem espaços antes e após o “=”.

```
a=world  
b="Hello World"
```

- ▶ As atribuições podem ser feitas de 3 formas, utilizando:

1.1 **Aspas** - interpreta o valor da variável:

```
nome="Paulo"  
variavel="Eu estou logado com o usuário $nome"  
echo $variavel # Saída = Eu estou logado com o usuário Paulo
```

1.2 **Apóstrofes (aspa simples)** - exibe o valor literal do conteúdo:

```
nome="José Paulo"  
variavel='Eu estou logado com o usuário $nome'  
echo $variavel # Saída = Eu estou logado com o usuário $nome
```

SOBRE OS SCRIPTS

OPERAÇÕES COM VARIÁVEIS

2. Acessar valor da variável:

- ▶ Atribuição de valor:

```
x=José
```

- ▶ Atribuir o valor de uma variável a outra:

```
y=$x
```

- ▶ Sem o "\$", você só estará atribuindo a string "x" a variável y:

```
y=x
```

- ▶ Concatenar o valor de uma variável com alguma string:

```
y=${x}Paulo
```

SOBRE OS SCRIPTS

OPERAÇÕES COM VARIÁVEIS



3. Imprimir na tela:

- ▶ O comando echo é o mais utilizado para escrever na tela textos ou valores de variáveis;
- ▶ Utilizar o echo sem nenhum sinal gráfico equivale a utilizá-lo seguido de aspas:

```
x=José  
echo Oi $x      # Saída = Oi José  
echo "Oi $x"    # Saída = Oi José
```

- ▶ Se você coloca o elemento após o echo entre apóstrofos (aspa simples), será exibido tal qual foi digitado:

```
x=José  
echo 'Oi $x'    # Saída = Oi $x
```


SOBRE OS SCRIPTS

OPERAÇÕES COM VARIÁVEIS



4. Ler uma variável do teclado:

- ▶ Utilizamos o comando read:

```
echo -n "Digite o nome: "  
read nome
```

- ▶ O parâmetro “-n” executa echo sem quebra de linha.

```
read -p "Digite o nome: " nome
```

- ▶ O parâmetro “-p” insere uma pausa até uma quebra de linha no fim do valor digitado.

SOBRE OS SCRIPTS

OPERAÇÕES COM VARIÁVEIS



5. Executar comandos:

- ▶ A execução de comandos acontece entre crases:
 - ▶ Seja na atribuição de valores a variáveis:

```
usuario=`whoami`  
echo 0 usuário logado: $usuario
```

- ▶ No momento de imprimir valores:

```
echo 0 usuário logado: `whoami`
```

ESTRUTURA CONDICIONAL - SE

INTRODUÇÃO



- ▶ Toda vez que você digita um comando no terminal, ele gera um status de saída, que vai de 0 a 255;
 - ▶ Quando o comando é executado com sucesso, esse código é 0.
 - ▶ O status do último comando digitado fica salvo numa variável de ambiente chamada “?”.
- ▶ Exemplo:

```
ls           # Após executar esse comando, vamos para o próximo
echo $?      # A saída será 0, para provar que o executou com sucesso
```

- ▶ O “0” no bash equivale a um “verdadeiro”, que tudo ocorreu bem;
 - ▶ Caso seja um comando que não existe, o retorno do echo será diferente de 0.
- ▶ Podemos tomar decisões baseadas nesse status, vamos supor que você fosse instalar vários pacotes, porém, eles precisam ser instalados numa ordem específica.

ESTRUTURA CONDICIONAL - SE

- ▶ Como já estudamos lógica de programação, o `if/else` é familiar;
- ▶ O `if/else` (`if then else` no shell) funciona de uma forma um pouco diferente da lógica tradicional;
 - ▶ Em outras linguagens, por exemplo, ao utilizar o `if` para checar uma condição, devemos indicar a condição que será testada;
 - ▶ Por exemplo: `0 == 0`.
 - ▶ Já no shell iremos testar a condição de um comando.
 - ▶ Vejamos um script que verifica se o comando foi executado corretamente:

```
#!/bin/bash
if ls; then           # if teste_lógico ; then
    echo "Funcionou"  # Valor se verdadeiro
else                  # else
    echo "Não funcionou" # Valor se falso
fi                    # Fim do if
```

- ▶ Obviamente que podemos fazer diversos testes lógicos, e vamos ver alguns operadores que podemos aplicar.

ESTRUTURA CONDICIONAL - SE

OPERADORES



► Operadores de comparação entre números:

```
#!/bin/bash
if [ 1 -eq 1 ]; then          # Verifica se são iguais
    echo "A condição é verdadeira"
else
    echo "A condição é falsa"
fi
```

- eq → igual a;
- ge → igual a e/ou maior que;
- gt → maior que;
- lt → menor que;
- le → igual a e/ou menor que;
- ne → diferente de.

ESTRUTURA CONDICIONAL - SE

OPERADORES



► Operadores de comparação entre strings:

```
#!/bin/bash
if [ `whoami` = 'root' ]; then      # Verifica se o usuário logado é o root
    echo 'Oi root';
fi
```

= → igual a;

!= → diferente de.

ESTRUTURA CONDICIONAL - SE

OPERADORES



► Associações entre condições:

```
#!/bin/bash
if [ `whoami` = 'severina' ] || [ `whoami` = 'maria' ]; then
    echo 'Olá usuária!';
else
    echo 'Olá usuário!';
fi
```

&& → operação AND lógico;

|| → operação OR lógico.

ESTRUTURA CONDICIONAL - SE

OPERADORES



► Operadores de teste de arquivo:

```
if [ -d /home ]; then
    echo 'É um diretório';
else
    echo 'Não é um diretório';
fi
```

- e** → a entrada existe;
- r** → a entrada pode ser lida;
- w** → a entrada pode ser escrita;
- x** → a entrada pode ser executada;
- s** → tem tamanho maior que zero;
- f** → é um arquivo;
- d** → é um diretório;
- 0** → o usuário logado é o proprietário da entrada;
- G** → grupo da entrada é o mesmo do proprietário;
- nt** → Verifica se um arquivo é mais novo que outro;

ESTRUTURA CONDICIONAL - SE

OPERADORES

► Operadores aritméticos:

```
if [ $((numero%2)) -eq '0' ]; then
    echo 'É par';
else
    echo 'É impar';
fi
```

+ → adição;

- → subtração;

* → multiplicação;

/ → divisão;

** → potenciação;

% → módulo (resto);

+= → incremento;

« ou » → deslocamento de bits;

«= ou »= → deslocamento e atribuição.

ESTRUTURA CONDICIONAL - SE

ESTRUTURA

- ▶ As condições testadas são status de saída da execução de comandos;
 - ▶ Lembrando: caso o status seja zero, condição é considerada verdadeira.
- ▶ Exemplo de `if/then`:

```
if [ $n1 -lt $n2 ]; then
    echo "$n1 é menor que $n2"
fi
```

- ▶ Existe um operador de condição denominado `test`, que também pode ser representado por `[condição]`, que retorna 0 quando a condição é verdadeira:

```
if test $n1 -lt $n2; then
    echo "$n1 é menor que $n2"
fi
```

ESTRUTURA CONDICIONAL - SE

ESTRUTURA

► Exemplo de if/then/else:

```
if cmp $file1 $file2 >/dev/null; then
    echo "Os arquivos são iguais"
else
    echo "Os arquivos são distintos"
```

- O comando `cmp` compara dois arquivos;
- A saída do comando `cmp` é uma linha contendo as diferenças entre os arquivos comparados.

► Exemplo de if/then/elif/else:

```
if [ $n1 -lt $n2 ]; then
    echo "$n1 < $n2"
elif [ $n1 -gt $n2 ]; then
    echo "$n1 > $n2"
else
    echo "$n1 = $n2"
```

ESTRUTURA CONDICIONAL - CASO



25

- Permite múltiplas opções de decisão:

```
case $nome in
    ola)                # case $variável in
                        # opção 1)
        echo "Olá você!" # Comandos para a opção 1
        ;;              # Fim dos comandos da opção 1
    xau)
        echo "Tchau!"
        ;;
    *)                  # Opção default, quando não existir uma
        echo "Não entendi."
        ;;
esac
```

ESTRUTURA DE REPETIÇÃO - PARA

- ▶ Exemplos de utilização do laço de repetição for:
 - ▶ Pode-se aplicar a repetição a uma lista de valores, strings, comandos, etc:

```
for planeta in Mercúrio Vênus Terra Marte    # Repetição para uma lista
do
    echo $planeta
done
```

- ▶ Incremento de um valor inicial até algum outro:

```
for i in {1..5}                                # Incremento de números
do
    echo "$i"                                    # Saída: 1, 2, 3, 4, 5
done
```

- ▶ A estrutura após o for variável in é: {Inicial..Final};
- ▶ Você pode fazer incrementos diferente de 1 com a seguinte estrutura: {Inicial..Final..Incremento}

ESTRUTURA DE REPETIÇÃO - PARA



- O comando `for` também pode ser representado na maneira “clássica”, que estamos habituados a ver noutras linguagens:

```
for ((i=1;i<10;i++))
do
    echo "$i"
done
```

ESTRUTURA DE REPETIÇÃO - ENQUANTO

- O laço de repetição `while` não difere muito das outras linguagens já estudadas:

```
#!/bin/bash
x=1
while [ $x -le 5 ]
do
    echo "$x"           # Saída: 1, 2, 3, 4, 5
    x=$(( $x + 1 ))
done
```

PARÂMETROS

Definição:

Parâmetro é um valor, proveniente de uma variável ou de uma expressão mais complexa, que pode ser passado para uma sub-rotina como nos scripts, utilizando os valores atribuídos nos parâmetros.

► Alguns parâmetros:

`$0` → nome do script;

`$n` → n-ésimo parâmetro, onde: $1 \leq n \leq 9$;

`$#` → quantidade de parâmetros;

`$*` → todos os parâmetros;

`$?` → status do último comando executado (“exit 1” retorna 1);

`$$` → número de processo (PID) do shell que executa o script.

► Ao chamar o script passa-se os parâmetros:

```
./script.sh UmParâmetro OutroParâmetro
```


PARÂMETROS

EXEMPLOS



Exemplo do script soma.sh:

```
#!/bin/bash  
echo "$1 + $2 = $(( $1 + $2 ))"
```

- Como chamar o script soma.sh já passando nos parâmetros os números a serem somados:

```
./soma.sh 5 4                      # Saída: 5 + 4 = 9
```

PARÂMETROS

EXEMPLOS

Outro exemplo utilizando outros tipos de operadores:

```
#!/bin/bash
echo "Nome do script: $0"
echo "Primeiro parâmetro passado: $1"
echo "Todos parâmetros passados: $*"
echo "Quantidade de parâmetros passados: $#"
```

► Chamando este script `NovoExemplo.sh`:

```
NovoExemplo.sh Bezerros Papangu Carnaval
```

► Saída deste script:

- Nome do script: `./NovoExemplo.sh`
- Primeiro parâmetro passado: Bezerros
- Todos parâmetros passados: Bezerros Papangu Carnaval

FUNÇÕES

Definição:

São blocos de código que podem ser utilizados em qualquer parte do script, quantas vezes for necessário. Para executar a função, basta fazer a chamada da função.

- ▶ A chamada da função se assemelha a um comando do bash;
 - ▶ Exemplo:

```
#!/bin/bash
quadrado ()
{
    echo "Digite um numero: "
    read entrada
    echo "$entrada ao quadrado é = $((entrada**2))"
}
echo "Programa iniciado..."
quadrado
```

- ▶ As variáveis definidas dentro das funções possuem escopo global e não são removidas após a

FUNÇÕES

EXEMPLO

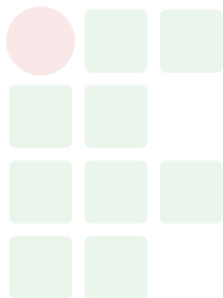


33

```
#!/bin/bash
somar ()
{
    resultado=$(( $1 + $2 ))
    return $resultado
}
echo "5 + 7 = "           # Escreve na tela o texto
somar 5 7                 # Chama a função, passando os parâmetros
echo $?                   # Exibe o retorno desta função
```

- ▶ \$1 e \$2 recebem os valores dos parâmetros da função;
- ▶ return indica o valor de retorno da função;
- ▶ \$? recebe o retorno da função.
 - ▶ Apenas valores numéricos inteiros podem ser retornados pelas funções.


REFERÊNCIAS

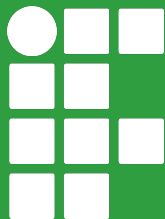


INSTITUTO
FEDERAL
Pernambuco

REFERÊNCIAS I



 TANENBAUM, Andrew S.; BOS, Herbert. **Sistemas Operacionais Modernos**. 4. ed. São Paulo: Pearson Education do Brasil, 2016.



INSTITUTO FEDERAL

Pernambuco