

Shell script: um guia básico

Shell script (sh) é uma linguagem de script usada em ambiente de linha de comando. Ela permite automatizar comandos de terminal em sistemas baseados em Unix, como Linux e MacOS.

Primeiros passos

Scripts são arquivos de texto que devem ser salvos com a extensão “.sh”. Por exemplo:

```
meu_script_shell.sh
```

Para executá-lo, o sistema deve ter a permissão de execução. Para isso, acesse o terminal, acesse o diretório em que o script se encontra (use o comando `cd`) e rode o comando:

```
chmod +x meu_script_shell.sh
```

Para executar o script, no terminal, digite:

```
./meu_script_shell.sh
```

Podemos ainda executar o programa, chamando no terminal:

```
sh meu_script_shell.sh
```

Comandos básicos em *Shell Script*

Agora, vamos criar um script básico que apenas exibe uma mensagem na tela. Crie um novo arquivo chamado ‘teste.sh’. Agora adicione as seguintes linhas de código:

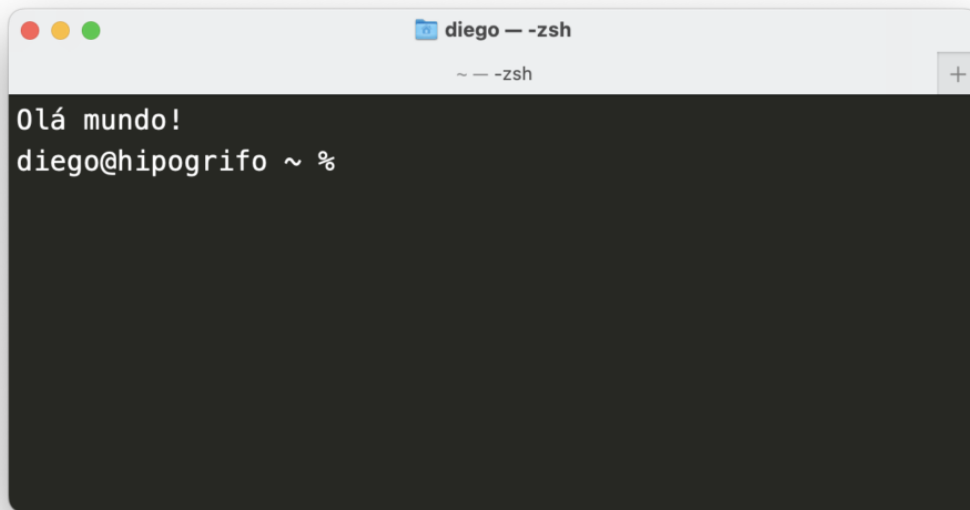
```
#!/bin/sh
# a linha acima declara o bin que executará o script

clear
# a linha acima executa um comando shell que limpa a tela

echo "Olá mundo!"
# a linha acima escreve a mensagem "Olá mundo"

# comentários são feitos com '#'
```

Ao executá-lo, veremos:

A terminal window titled "diego — -zsh" with a subtitle "~ — -zsh". The window shows the output of a script: "Olá mundo!" followed by the prompt "diego@hipogrifo ~ %".

```
Olá mundo!  
diego@hipogrifo ~ %
```

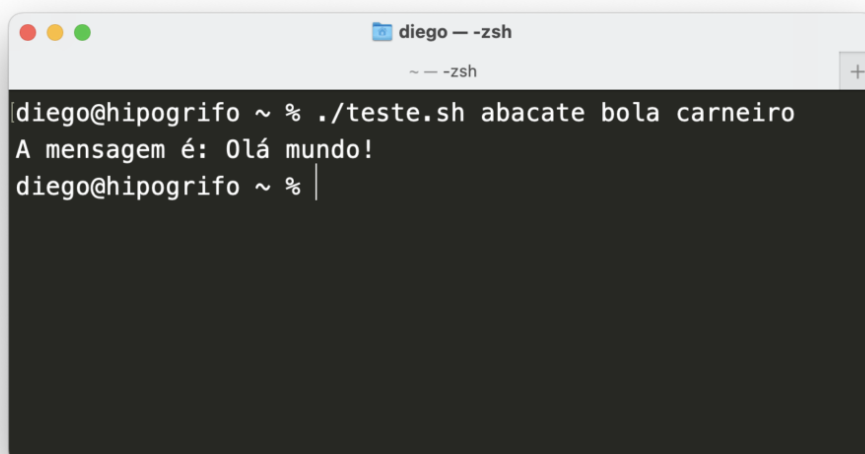
O script foi executado com “./teste.sh”. Note que essa mensagem foi apagada da tela pelo comando de terminal “clear”.

Criando variáveis em *Shell Script*

Podemos criar uma variável apenas inserindo um nome e atribuindo um valor com o operador “=”. Note que não podemos ter espaços entre o nome da variável, o operador “=” e o valor atribuído. Veja:

```
#!/bin/sh  
  
mensagem="Olá mundo!"  
  
echo "A mensagem é: $mensagem"
```

Observe o resultado:

A terminal window titled "diego — -zsh" with a subtitle "~ — -zsh". The window shows the execution of the script: "diego@hipogrifo ~ % ./teste.sh abacate bola carneiro", followed by the output "A mensagem é: Olá mundo!", and then the prompt "diego@hipogrifo ~ %".

```
diego@hipogrifo ~ % ./teste.sh abacate bola carneiro  
A mensagem é: Olá mundo!  
diego@hipogrifo ~ %
```

Comando executado no terminal. Neste caso, executamos um script chamado 'teste.sh'.

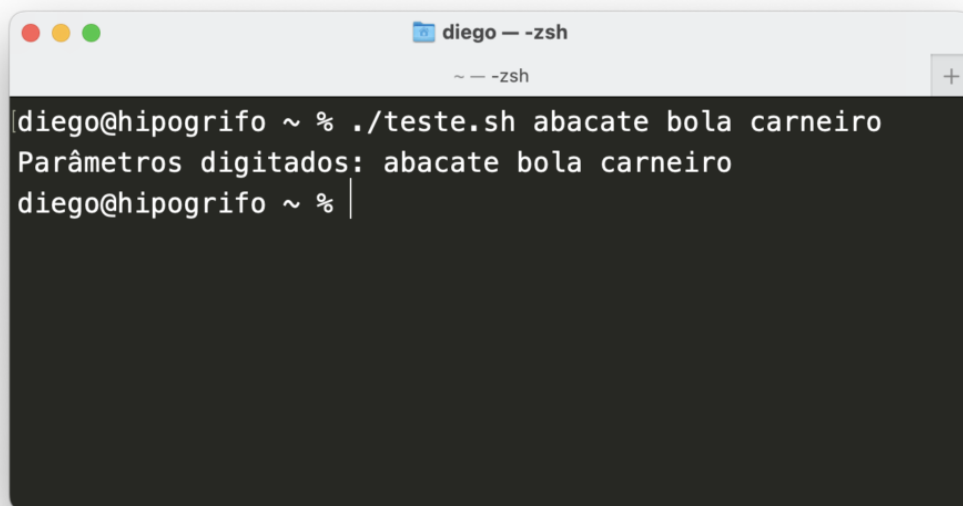
Note que para chamar as variáveis, precisamos declarar um cifrão "\$" antes de seu nome. Entretanto, esse cifrão não é necessário na hora de declará-la.

Recebendo argumentos

Podemos receber argumentos enviados na chamada do programa usando as variáveis \$1, \$2, \$3, ... \$9:

```
#!/bin/sh

echo "Parâmetros digitados: $1 $2 $3"
```



Neste exemplo, três parâmetros foram passados na chamada do programa.

Estruturas condicionais

Estruturas condicionais podem ser aplicadas usando as estruturas `if`, `elif` e `else`. Veja a sintaxe:

```
#!/bin/sh

if [ condicao ]; then
    echo '...'
elif [ outra_condicao ]; then
    echo '...'
else
    echo '...'
fi
```

Condições são aplicadas entre colchetes `[]`. Após cada condição, devemos adicionar `;` `then`. Além disso, ao final precisamos adicionar o comando `fi`. Note que para avaliarmos uma condição, precisamos primeiro conhecer os operadores relacionais, lógicos ou aritméticos.

Operadores relacionais

Para comparar valores, precisamos utilizar os operadores relacionais. São eles:

Operador	Definição
<code>-eq</code>	Igual
<code>-ne</code>	Diferente
<code>-gt</code>	Maior que
<code>-lt</code>	Menor que
<code>-ge</code>	Maior ou igual que
<code>-le</code>	Menor ou igual que

Operadores comparativos em Shell Script.

Fonte: adaptado de <https://mange.ifrn.edu.br/shell-script-wikipedia/20-operadores-relacionais.html>

Observe um exemplo com o operador `-gt` (*greater than*/maior que):

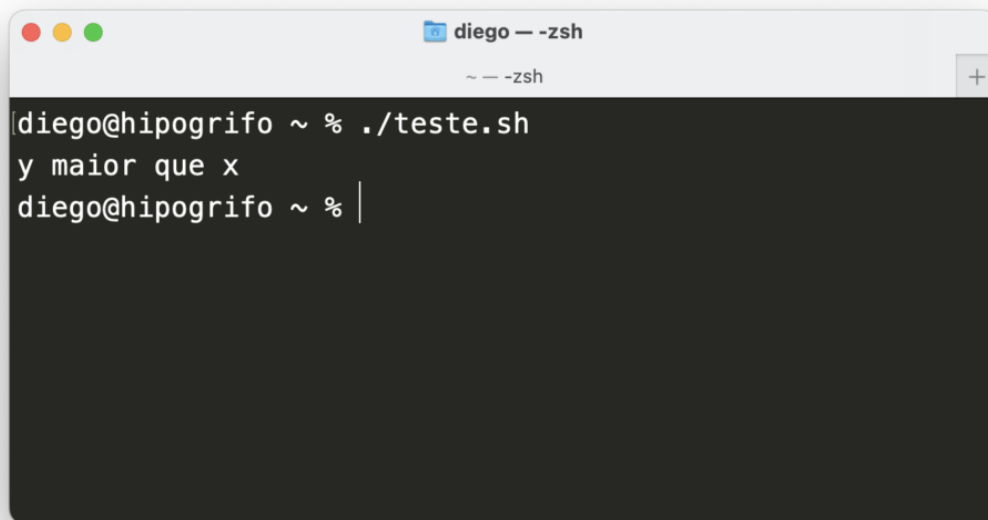
```
#!/bin/sh

x=10
y=20

# verifica se x é maior que y
if [ $x -gt $y ]; then
    echo 'x maior que y'
fi

# verifica se y é maior que x
if [ $y -gt $x ]; then
    echo 'y maior que x'
fi
```

Veja que será exibido:

A terminal window titled 'diego — -zsh' with a subtitle '~ — -zsh'. The prompt is 'diego@hipogrifo ~ %'. The user has entered './teste.sh'. The output of the script is 'y maior que x'. The prompt is now 'diego@hipogrifo ~ % |'.

```
diego@hipogrifo ~ % ./teste.sh
y maior que x
diego@hipogrifo ~ % |
```

Note que poderíamos, fazer isso usando uma única estrutura condicional, usando `elif`:

```
#!/bin/sh

x=10
y=20

# verifica se x é maior que y
if [ $x -gt $y ]; then
    echo 'x maior que y'
elif [ $y -gt $x ]; then
    echo 'y maior que x'
fi
```

Podemos ainda adicionar uma condição final, caso nenhuma outra condição seja verdadeira:

```
#!/bin/sh

x=10
y=10

# verifica se x é maior que y
if [ $x -gt $y ]; then
    echo 'x maior que y'
elif [ $y -gt $x ]; then
    echo 'y maior que x'
else
    echo "x é igual a y"
fi
```

```
diego@hipogrifo ~ % ./teste.sh
x é igual a y
diego@hipogrifo ~ %
```

Neste caso, x é igual a y.

Operadores lógicos

Já os operadores lógicos permitem realizar múltiplas comparações. Eles podem ser:

Operadores	Descrição	Exemplos
!	Diferente	<pre>[! \$x = 10]</pre> Verifica se o valor de x é diferente de 10.
-o	ou (or)	<pre>[\$x = 10 -o \$x = 20]</pre> Verifica se o valor de x é igual a 10 ou 20
-a	e (and)	<pre>[\$x -gt 0 -a \$x -lt 10]</pre> Verifica se o valor de x é maior que 0 e menor que 10.

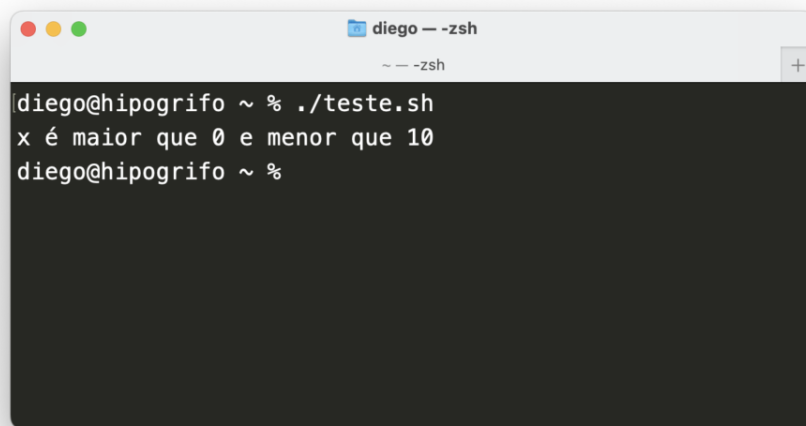
Operadores relacionais. Fonte: adaptado de <https://mange.ifrn.edu.br/shell-script-wikipedia/18-operadores-booleanos.html>

Observe um exemplo:

```
#!/bin/sh

x=6

if [ $x -gt 0 -a $x -lt 10 ]; then
    echo 'x é maior que 0 e menor que 10'
fi
```

A terminal window titled 'diego - zsh' with a subtitle '~ - zsh'. The prompt is 'diego@hipogrifo ~ %'. The user enters './teste.sh'. The output is 'x é maior que 0 e menor que 10'. The prompt returns to 'diego@hipogrifo ~ %'.

Operadores aritméticos

Operadores aritméticos permitem realizar operações matemáticas em Shell Script. Eles podem ser:

Operador	Descrição	Exemplo
+	Adição	<code>\$(expr 10 + 10)</code> # 20
-	Subtração	<code>\$(expr 120 - 20)</code> # 100
*	Multiplicação	<code>\$(expr 10 * 7)</code> # 70
/	Divisão	<code>\$(expr 10 / 2)</code> # 5
%	Módulo	<code>\$(expr 11 % 2)</code> # 1
=	Igualdade	Note que: <code>x=10</code> atribui o valor 10 à variável <code>x</code> . Enquanto: <code>[\$x = 10]</code> , verifica se a variável <code>\$x</code> é igual a 10
!=	Diferente	<code>[\$x != 2]</code> A variável <code>x</code> é diferente de 2?

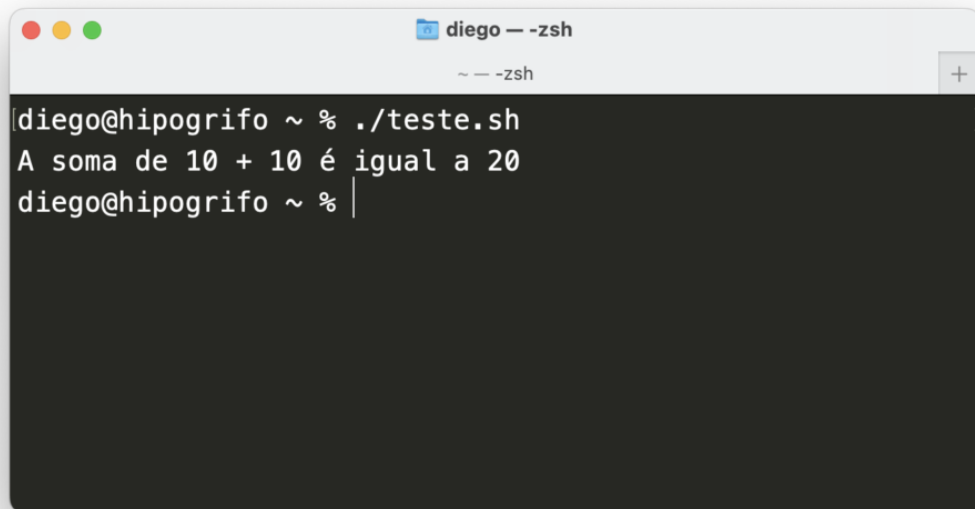
Operadores aritméticos. Fonte: adaptado de <https://mange.ifrn.edu.br/shell-script-wikipedia/17-operadores-aritmeticos.html>

Observe como podemos fazer uma simples soma:

```
#!/bin/sh

x=10
y=10
soma=$(expr $x + $y )

echo "A soma de $x + $y é igual a $soma"
```

A terminal window titled 'diego -- zsh' with a subtitle '~ -- zsh'. The prompt is 'diego@hipogrifo ~ %'. The user enters './teste.sh'. The output is 'A soma de 10 + 10 é igual a 20'. The prompt returns to 'diego@hipogrifo ~ %' with a cursor.

```
diego@hipogrifo ~ % ./teste.sh
A soma de 10 + 10 é igual a 20
diego@hipogrifo ~ %
```

Estruturas de repetição

Há duas principais estruturas de repetição: `for` e `while`. Observe um exemplo usando o comando `for`:

```
#!/bin/sh

for ((i=1; i<5; i++))
do
    echo $i
done
```

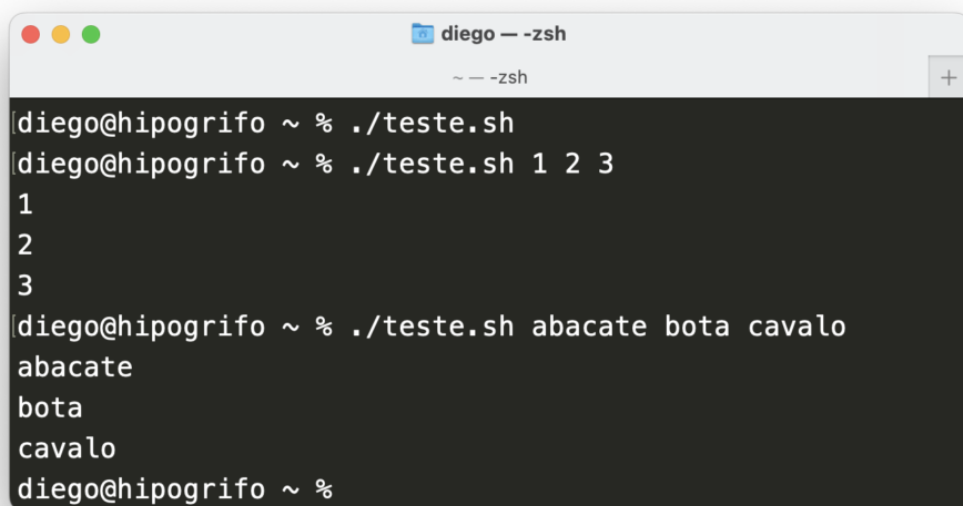



```
diego@hipogrifo ~ % ./teste.sh
1
2
3
4
diego@hipogrifo ~ %
```

Veja como podemos usar for para pegar todos os argumentos enviados na chamada do script:

```
#!/bin/sh

# para cada argumento como $i
for i in "$@"
do
    echo $i
done
```



```
diego@hipogrifo ~ % ./teste.sh
diego@hipogrifo ~ % ./teste.sh 1 2 3
1
2
3
diego@hipogrifo ~ % ./teste.sh abacate bota cavalo
abacate
bota
cavalo
diego@hipogrifo ~ %
```

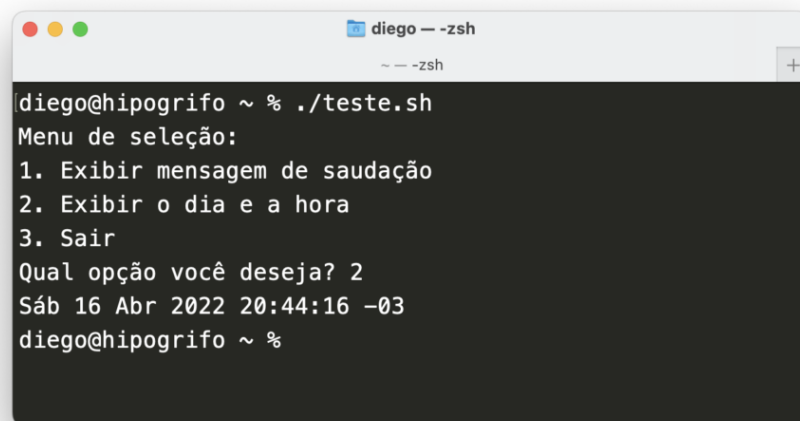
Três chamadas distintas do script. Observe os argumentos impressos.

Observe como podemos usar o comando `while` para criar um menu de seleção:

```
#!/bin/sh

echo "Menu de seleção:"
echo "1. Exibir mensagem de saudação"
echo "2. Exibir o dia e a hora"
echo "3. Sair"

while read -p "Qual opção você deseja? " entrada
do
    if [ $entrada -eq 1 ]; then
        echo "Bem-vindo(a)!"
        break
    elif [ $entrada -eq 2 ]; then
        date
        break
    else
        echo "Até mais!"
        break
    fi
done
```



A screenshot of a terminal window titled "diego — zsh". The prompt is "diego@hipogrifo ~ %". The user has entered the command ". /teste.sh". The script output is as follows:

```
Menu de seleção:
1. Exibir mensagem de saudação
2. Exibir o dia e a hora
3. Sair
Qual opção você deseja? 2
Sáb 16 Abr 2022 20:44:16 -03
diego@hipogrifo ~ %
```

Automatizando comandos com Shell Script

Podemos usar qualquer comando de terminal dentro de um arquivo “.sh”. Por isso, podemos utilizar esse tipo de script para automatizar tarefas administrativas ou até mesmo execução múltipla de scripts.

Lendo todos os arquivos de um diretório

Podemos ler todos os arquivos de um diretório usando o comando “ls”. Veja:

```
#!/bin/sh

arquivos=$(ls)

echo $arquivos
```

Este comando irá listar todo o conteúdo do diretório atual.

Agora, vamos fazer algo mais avançado. Vamos concatenar vários comandos em sequência.

```
#!/bin/sh
```

```
arquivos=$(mkdir teste && cd teste && echo 'teste' > exemplo.txt  
&& ls)
```

```
echo $arquivos
```

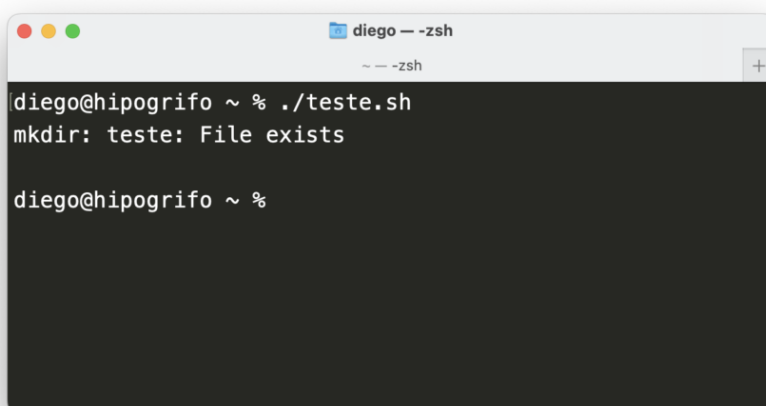
Acima, estamos criando uma pasta chamada teste com o comando `mkdir`, depois entramos nessa pasta e escrevemos a palavra “teste” que será salva em um arquivo chamado “exemplo.txt”. Por fim, executamos o comando “ls” para ver se o arquivo realmente foi criado.

Note que ao executar o código, você verá apenas isto:

A terminal window titled 'diego -- zsh' with a subtitle '~ -- zsh'. The prompt is 'diego@hipogrifo ~ %'. The user enters './teste.sh', and the terminal outputs 'exemplo.txt' on the next line. The prompt returns to 'diego@hipogrifo ~ %' with a cursor.

```
diego@hipogrifo ~ % ./teste.sh  
exemplo.txt  
diego@hipogrifo ~ %
```

De fato, o arquivo desejado foi criado. Podemos perceber isso, executando novamente o código (o sistema retornará um erro, dizendo que a pasta já existe).

A terminal window titled 'diego -- zsh' with a subtitle '~ -- zsh'. The prompt is 'diego@hipogrifo ~ %'. The user enters './teste.sh', and the terminal outputs 'mkdir: teste: File exists' on the next line. The prompt returns to 'diego@hipogrifo ~ %' with a cursor.

```
diego@hipogrifo ~ % ./teste.sh  
mkdir: teste: File exists  
  
diego@hipogrifo ~ %
```

Lendo vários arquivos de texto usando shell script

Agora, vamos criar vários arquivos de texto em um diretório e lê-los usando shell script:

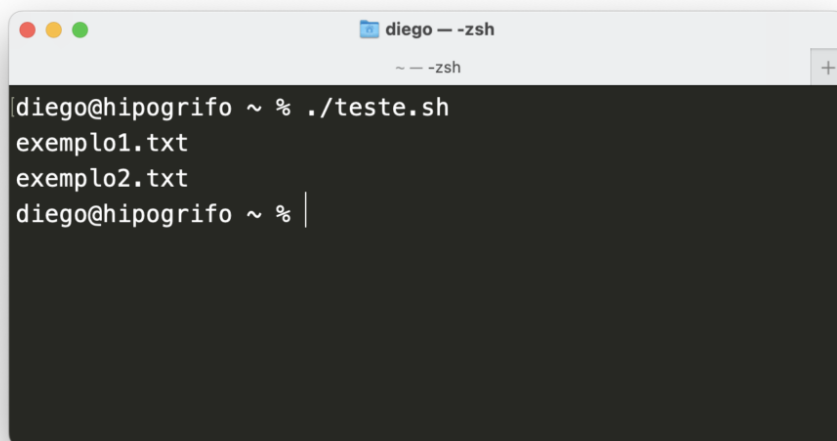
```
#!/bin/sh

echo 'teste1' > exemplo1.txt
echo 'teste2' > exemplo2.txt

arquivos=$(ls *txt)

for i in $arquivos
do
    echo $i
done
```

Note que o operador “>” joga o resultado impresso na tela para um arquivo.



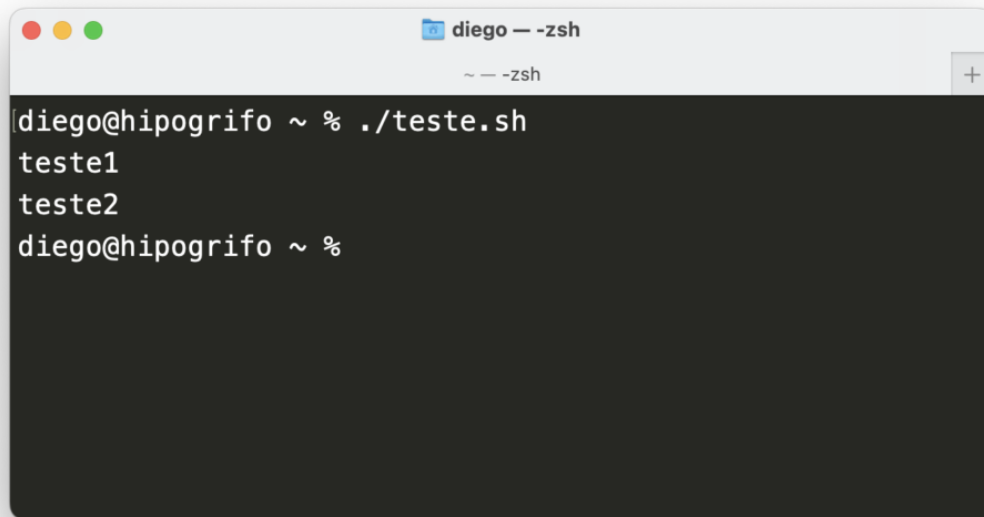
Conseguimos detectar os arquivos, mas como podemos fazer para lê-los? Basta usar o comando “cat” no lugar de “echo”.

```
#!/bin/sh

echo 'teste1' > exemplo1.txt
echo 'teste2' > exemplo2.txt

arquivos=$(ls *txt)

for i in $arquivos
do
    cat $i
done
```

A terminal window titled 'diego - zsh' with a subtitle '~ - zsh'. The prompt is 'diego@hipogrifo ~ %'. The user enters './teste.sh', followed by 'teste1' and 'teste2'. The prompt returns to 'diego@hipogrifo ~ %'.

```
diego@hipogrifo ~ % ./teste.sh
teste1
teste2
diego@hipogrifo ~ %
```

Podemos ainda gravá-lo em um único arquivo, usando o operador “>>”, que joga o resultado da execução para um arquivo (modo incremental).

No exemplo a seguir, vamos ler todos os arquivos com cat e gravá-los em um outro arquivo chamado “todos_exemplos.txt”:

```
#!/bin/sh

echo 'teste1' > exemplo1.txt
echo 'teste2' > exemplo2.txt

arquivos=$(ls *txt)

for i in $arquivos
do
    cat $i >> todos_exemplos.txt
done
```

Conclusões

Aqui apresentamos alguns exemplos de comandos SH. O Shell Script pode ser uma ferramenta fundamental para usuários de sistemas baseados em Unix que desejam automatizar tarefas no sistema operacional.

Para saber mais, execute no terminal `sh --help`. Até mais!

Créditos do conteúdo:

Fonte: Diego Mariano (CC BY-NC).