

Laboratório de SO

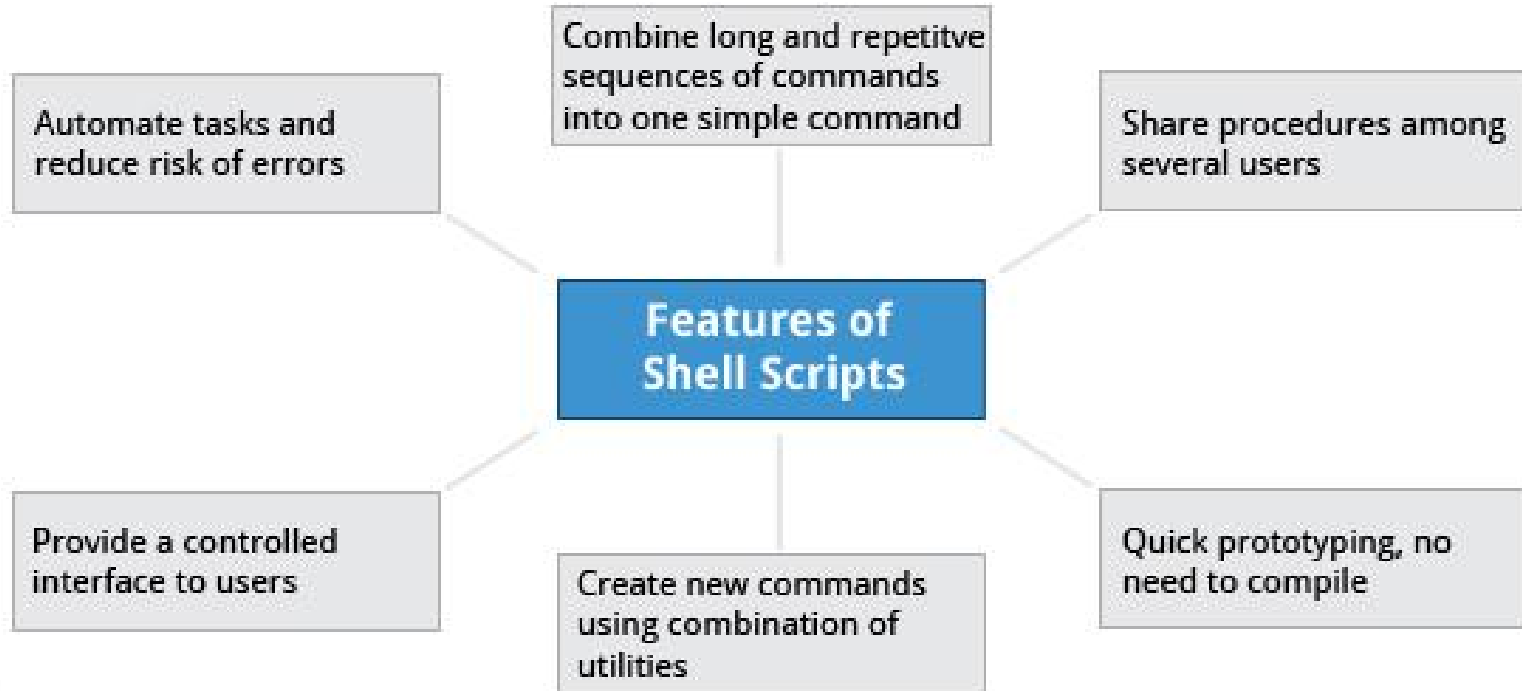
introdução ao shell script

Introdução ao Scripting

Suponha que você deseja procurar um nome de arquivo, verifique se o arquivo associado existe e, em seguida, responda de acordo, exibindo uma mensagem confirmando ou não a existência do arquivo. Se você só precisa fazer isso uma vez, basta digitar uma seqüência de comandos em um terminal. No entanto, se você precisar fazer isso várias vezes, a **automação** é o caminho a percorrer.

Para automatizar conjuntos de comandos, você precisará aprender a escrever **scripts de shell**, o bash. O gráfico ilustra vários dos benefícios da implantação de scripts.

Introdução ao Scripting



Vamos escrever um script bash simples que exibe uma mensagem de uma linha na tela. Qualquer tipo

```
$ cat > olamundo.sh
#!/bin/bash
echo "Olá Mundo!!!"
```

Em seguida, digite `chmod +x olamundo.sh` para tornar o arquivo executável por todos os usuários.

Você pode então executar o script digitando `./olamundo.sh` ou fazendo:

```
$ bash olamundo.sh
Olá Mundo!!!
```

.

Exemplo de leitura de variáveis

Agora, vamos ver como criar um exemplo mais interativo usando um script **bash**. O usuário será solicitado a digitar um valor, que é exibido na tela. O valor é armazenado em uma variável temporária, **NOME**. Podemos referenciar o valor de uma variável de shell usando um \$ na frente do nome da variável, como **\$NOME**. Para criar este script, você precisa criar um arquivo chamado **mostranome.sh**:

```
#!/bin/bash
# leitura interativa de uma variável
echo "Entre com seu nome:"
read NOME
# mostra o valor lido
echo "O nome dado foi: $NOME"
```

exercícios

Escreva um script que (criapasta.sh):

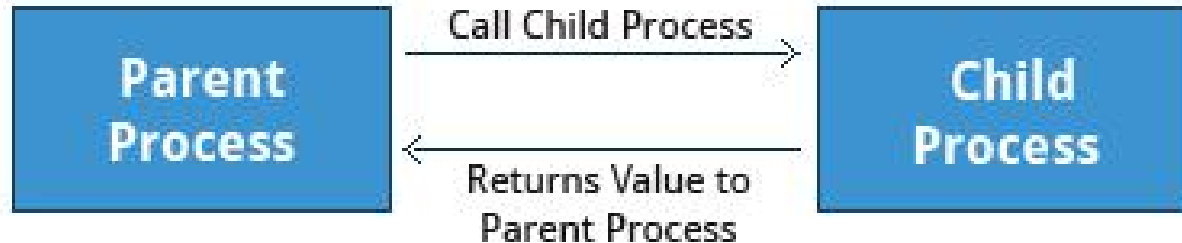
1 - pergunte seu nome e :

2 - crie uma pasta com o nome inserido e...

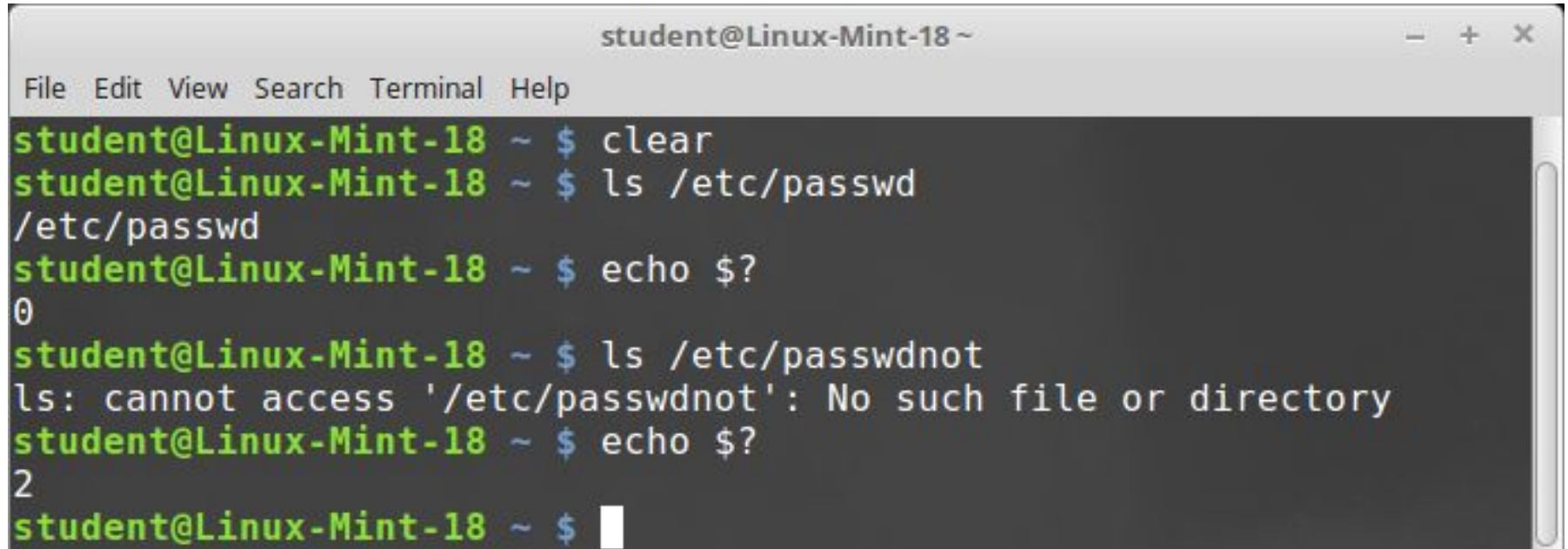
3 - dentro da pasta crie 3 arquivos com extensão .txt e inicia com o nome inserido + a sequencia 1, 2 e 3. ex.: alex1.txt, alex2.txt e alex3.txt

Valores de Retorno

Todos os scripts de shell geram um valor de retorno ao finalizar a execução, o valor pode ser definido com a instrução **exit**. Valores de retorno permitem que um processo monitore o estado de saída de outro processo, geralmente em uma relação pai-filho. Isso ajuda a determinar como este processo terminou e tomar quaisquer medidas necessárias, dependentes de sucesso ou fracasso.



Visualizando os valores de retorno

A terminal window titled 'student@Linux-Mint-18 ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows a sequence of commands and their return values. The first command 'clear' returns 0. The second command 'ls /etc/passwd' returns 0. The third command 'echo \$?' returns 0. The fourth command 'ls /etc/passwdnot' returns an error message: 'ls: cannot access '/etc/passwdnot': No such file or directory'. The fifth command 'echo \$?' returns 2. The sixth command is a prompt with a cursor.

```
student@Linux-Mint-18 ~ $ clear
student@Linux-Mint-18 ~ $ ls /etc/passwd
/etc/passwd
student@Linux-Mint-18 ~ $ echo $?
0
student@Linux-Mint-18 ~ $ ls /etc/passwdnot
ls: cannot access '/etc/passwdnot': No such file or directory
student@Linux-Mint-18 ~ $ echo $?
2
student@Linux-Mint-18 ~ $
```


exercícios

Qual comando é usado para tornar o **some_script.sh** executável?

☐ `chmod a some_script.sh`

☐ `chmod +x some_script.sh`

☐ `./some_script.sh some_command`

☐ `cat ./somescript.sh`

exercícios

Escreva um script que (valoretorno.sh):

Fazer um **ls** para um arquivo inexistente e, em seguida, exibe o status de saída resultante.

Criar um arquivo e fazer um **ls** para ele e, em seguida, exibe novamente o status de saída resultante.

exercícios

Escreva um script que:

1 - pergunte seu nome e :

2 - crie uma pasta com o nome inserido e...

3 - dentro da pasta crie 3 arquivos com extensão .txt e inicia com o nome inserido + a sequencia 1, 2 e 3. ex.: alex1.txt, alex2.txt e alex3.txt

Parâmetros de Script

Os usuários geralmente precisam passar valores de parâmetro para um script, como um nome de arquivo, data, etc. Os scripts terão caminhos diferentes ou chegarão a valores diferentes de acordo com os parâmetros (argumentos de comando) passados a eles. Esses valores podem ser texto ou números como em:

```
$ ./script.sh /tmp
```

```
$ ./script.sh 100 200
```

Dentro de um script, o parâmetro ou um argumento é representado com um \$ e um número ou caractere especial. A tabela lista alguns desses parâmetros.

Parâmetros de Script

Parameter	Meaning
\$0	Script name
\$1	First parameter
\$2, \$3, etc.	Second, third parameter, etc.
\$*	All parameters
\$#	Number of arguments

Usando os parâmetros

De acordo com a saída abaixo, crie o script executável com **chmod + x param.sh**. Em seguida, execute o script dando vários argumentos, como mostrado.

linha de comando: **./param.sh um dois tres quatro cinco**

saída na tela:

\$0 imprime, o nome do script: **param.sh**

\$1 imprime, o primeiro parâmetro: **um**

\$2 imprime, o segundo parâmetro: **dois**

\$3 imprime, o terceiro parâmetro: **três**

\$* Imprime, todos os parâmetros: **um dois três quatro cinco**

exercício

Crie um script”:

envia_github “mensagem”

Que deverá enviar todos os arquivos modificados para o github

Substituição de Comando

Às vezes, você pode precisar substituir o resultado de um comando como uma parte de outro comando. Pode ser feito de duas maneiras:

Ao incluir o comando interno com aspas simples(`)

Ao incluir o comando interno em \$ ()

Não importa o método, o comando mais interno será executado em um ambiente de shell recém-lançado e a saída padrão do shell será inserida onde a substituição de comando foi feita.

Substituição de Comando

Praticamente qualquer comando pode ser executado desta forma. Ambos os métodos permitem a substituição de comandos; No entanto, o método \$ () permite o aninhamento de comandos. Novos scripts devem sempre usar este método mais moderno. Por exemplo:

```
$ cat $(grep -l "linux" *)
```

```
$ cat `grep -l "linux" *`
```

No exemplo acima, a saída do comando "grep -l "linux" torna-se o argumento para o comando ls

exercício

qual a diferença entre os 2 comandos

echo ls

echo \$(ls)

exercício

Faça um script chamado **soma.sh** que recebe 2 parâmetros números de entrada e imprime a soma desses 2 números. Exemplo:

```
> bash soma.sh 4 5
```

```
soma = 9
```

funções

Uma função é um bloco de código que implementa um conjunto de operações. As funções são úteis para executar procedimentos várias vezes talvez com variáveis de entrada. As funções também são chamadas frequentemente sub-rotinas. Usar funções em scripts requer duas etapas:

1. Declarar uma função
2. Chamando uma função.

A declaração de função requer um nome que é usado para invocá-lo. A sintaxe adequada é:

```
function_name () {  
    comando...  
}
```

funções

O que será impresso após a execução do programa (minhaversao.sh) a seguir

```
1 #!/bin/bash
2 minhaversao()
3 {
4     echo minha distribuição preferida do Linux é $1
5 }
6
7 minhaversao "Ubuntu"
8 minhaversao "Fedora"
9
```