VIII. Scripts em Shell

1. Introdução

Assim como podemos executar comandos diretamente na linha de comando em shell, no Linux, podemos criar arquivos do tipo texto com as sequências de comandos a serem executados, como se fossem digitados diretamente na linha de comando.
Esses arquivos geralmente são chamados de scripts, ou shell scripts.

Isso permite automatizar tarefas rotineiras aumentando a produtividade do administrador do sistema operacional, do desenvolvedor ou mesmo do usuário final.

Exemplo: Se todo dia precisamos realizar o backup de uma pasta do sistema, e remover arquivos temporários antigos, podemos criar um script com os comandos para realizar a tarefa.
Para isso, basta editar um arquivo texto e incluir os comandos necessários, e atribuir permissão de execução.
Suponha que tenhamos uma pasta chamada SO, com os arquivos .txt criados, e diariamente copiamos esses arquivos para dentro de uma pasta bkp, e os que estavam dentro da pasta bkp são movidos para outra pasta bkp.ant. E os que estavam dentro da bkp.ant são removidos. Ou seja, queremos backup dos dois dias anteriores.
Para isso, dentro da pasta SO, criamos um arquivo chamado `backup`, e nele incluímos os seguintes comandos

```
cd ~/SO
rm -r bkp.ant
mv bkp bkp.ant
mkdir bkp
cp *.txt bkp/.
```

Observe que o script entra na pasta `SO`, remove a pasta `bkp.ant`, renomeia a pasta `bkp` para `bkp.ant`, recria a pasta `bkp` e então copia os arquivos `.txt` para dentro da pasta `bkp`.
Para executar isso todo dia, são necessários 5 comandos.
Uma vez que foi atribuída permissão de execução para o script, basta executar o script.
Isso pode ser feito de duas formas: digitando o nome do script (com o caminho até ele, se não estiver no PATH (`./backup`)), ou acionando outro shell para executá-lo (`bash backup`).

A execução de shell scripts pode ser feita com a **opção de depuração**, para auxiliar a encontrar eventuais erros. Nesse caso cada comando executado é listado na tela, precedido por um sinal `+`.
Para acionar a execução com depuração, usa-se a opção `-x`.
Exemplo, para rodar o script `backup` com a opção de depuração, executa-se `bash -x backup`.

2. Parâmetros

É frequente precisarmos informar parâmetros para um script, assim como informamos parâmetros para os comandos do shell.

Dentro do script, os parâmetros são reconhecidos pelo caracter ‘$’ seguido pela posição em que foram informados.
O primeiro parâmetro é $1, o segundo parâmetro é $2, e assim sucessivamente.

Por exemplo, se quisermos fazer um script que soma dois números e apresenta o resultado, podemos criar um script chamado `soma`, com o seguinte conteúdo.

```
let result=$1+$2
echo $result
```

E para chamar o script somando 6 com 7, depois de atribuir permissão de execução (`chmod 755 soma`) basta executar o comando `./soma 6 7`

Para evitar erros, é interessante verificar se o usuário está passando o número correto de parâmetros. Isso é feito testando o parâmetro especial `$#`.
Por exemplo, podemos melhorar o script anterior testando se o usuário passou dois parâmetros

```
if [ $# -eq 2 ]
then
   let result=$1+$2
   echo $result
else
   echo "Número de parâmetros inválido. Informe dois números"
fi
```

Em alguns casos, pode ser necessário fazer scripts com número variável de parâmetros.
Nesse caso, o processamento de parâmetros pode ser feito um a um.
Em geral se faz um laço de repetição, enquanto houver parâmetros.
O primeiro parâmetro é processado em seguida desprezado pelo comando `shift`
Um exemplo pode ser encontrado na seção 5, sobre o laço `while`.

3. Testes – if – then – else

A linguagem script possui muitos recursos de programação, incluindo os principais recursos de uma programação estruturada, sendo razoavelmente poderosa.
A sintaxe do `if` tem vários recursos e pode ser usada de acordo com o seguinte exemplo:

```
if [ <condição> ]
then
   <comando 1>
        :
   <comando n>
fi
```

ou

```
if [ <condição> ]
then
   <comando 1>
        :
   <comando n>
else
   <comando 1>
        :
   <comando n>
fi
```

Algumas opções de **teste de arquivo** são:

`-e <nome>` testa se um arquivo <nome> existe
`-f <nome>` testa se um arquivo <nome> existe e é regular
`-d <nome>` testa se um arquivo <nome> existe e é um diretório

Uma relação completa de opções de teste disponíveis pode ser obtida em `man test`

Exemplo: Para testar se um arquivo informado por parâmetro existe, podemos usar o script

```
# Verifica se um arquivo informado por parâmetro existe
# $1 - Nome de arquivo informado
if [ -e $1 ]
then
   echo "O arquivo $1 existe"
else
   echo "O arquivo $1 não existe"
fi
```

Para testar se um arquivo informado por parâmetro existe e é um arquivo regular, podemos usar o script

```
# Verifica se o arquivo informado por parâmetro existe e é regular
# $1 - Nome de arquivo informado
if [ -e $1 ]
then
   if [ -f $1 ]
   then
      echo "O arquivo $1 é regular"
   else
      echo "O arquivo $1 não é regular"
   fi
else
   echo "O arquivo $1 não existe"
fi
```

Uma versão melhorada desse script pode testar se foi informado um nome de arquivo por parâmetro

```
# Verifica se o arquivo informado por parâmetro existe e é regular
# $1 - Nome de arquivo informado
if [ $# -eq 1 ]
then
   if [ -e $1 ]
   then
      if [ -f $1 ]
      then
         echo "O arquivo $1 é regular"
      else
         echo "O arquivo $1 não é regular"
      fi
   else
      echo "O arquivo $1 não existe"
   fi
else
   echo "Erro: Nome de arquivo não informado"
fi
```

O `if` também tem testes de **comparação numérica**. Mas não pode usar os símbolos < e >, porque eles são interpretados como redirecionamento de entrada e saída para aquivos. Então são usados os seguintes códigos:

`-gt` = Maior que (*greater than*)
`-lt` = Menor que (*less than*)
`-eq` = Igual (*equal*)
`-ge` = Maior ou igual (*greater or equal*)
`-le` = Menor ou igual (*less or equal*)

Exemplo: para fazer um script que compara dois números, informando se o primeiro é maior, igual ou menor que o segundo, podemos fazer:

```
if [ $# -eq 2 ] # testa se o número de parâmetros é igual a 2
then
   if [ $1 -gt $2 ] # testa se o primeiro parâmetro é maior que o segundo
   then
      echo "$1 eh maior que $2"
   else
      if [ $1 -eq $2 ] # testa se o primeiro parâmetro é igual que o segundo
      then
         echo "Os dois números são iguais"
      else
         echo "$1 é menor que $2"
      fi
   fi
else
   echo "Informe dois parâmetros"
fi
```

A comparação de strings pode ser feita com os testes “=” (igual) e “!=” (diferente).

4. Laços – for

A estrutura de repetição `for` existe no shell e é bastante poderosa, podendo interagir com saídas de outros comandos.
A sintaxe é:

```
for <variável> in <conjunto de valores>
do
   <comando 1>
      :
   <comando n>
done
```

A variável informada no laço vai assumir todos os valores informados no conjunto de valores.
Por exemplo, se queremos imprimir os valores de 1 a 10, podemos fazer:

```
for i in 1 2 3 4 5 6 7 8 9 10
do
    echo $i
done
```

ou, especificamente para o bash, pode ser escrito como:

```
for i in {1..10}
do
    echo $i
done
```

ou ainda como:

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

Os valores assumidos pela variável não precisam ser numéricos. Podem ser string também. Exemplo: Suponha que existam os diretórios `bkp.segunda`, `bkp.terca`, ... `bkp.sexta`, para armazenar os backups de cada dia útil da semana. O script abaixo lista na tela o conteúdo de todas as pastas.

```
for dia in segunda terca quarta quinta sexta
do
    echo "Backup de $dia"
    ls -l bkp.$dia
done
```

Mas o mais interessante deste comando é que podemos fazer ele usar a saída de outros comandos. Por exemplo, se queremos fazer algum processamento (nesse exemplo, listar na tela) com os arquivos `*.txt` de uma pasta informada (primeiro parâmetro - `$1`), podemos usar o comando

```
for i in `ls $1/*.txt`
do
    echo $i
done
```

Aqui é importante destacar a função do sinal crase (`` ` ``). O comando que estiver entre as crases é executado, e o seu resultado retorna para o script, nesse caso, para o comando `for`, e a variável `i` vai assumir cada um dos strings retornados pelo comando `ls` (nesse caso, o nomes dos arquivos da pasta `$1` que terminam com `.txt`).