# Introdução em Shell Script

Guilherme Magalhães Gall
gmgall@gmail.com

# O que é um Shell Script?

- Arquivo que guarda uma sequência de comandos e pode ser executado quando necessário.
- Os comandos de um script são exatamente os mesmos que os que se digita no prompt.
- Existem estruturas de decisão (`if`, `case`) e de repetição (`while`, `for`, `until`) assim como em outras linguagens.

# Primeiro script

- Crie um arquivo `hello.sh` cuja primeira linha seja `#!/bin/bash`
- Dê permissão de execução ao arquivo.
- Execute o script com `./hello.sh`

```
$ vi hello.sh
$ chmod +x hello.sh
$ ./hello.sh
Olá mundo!
```

# Primeiro script

- Para executar scripts que estão fora do PATH é necessário passar o caminho completo para o script, por isso o ./ na frente do nome.
- Lembra que o meta-caractere . (ponto) é expandido para o diretório atual?

# Exit status

- Todo processo no Linux retorna um código de retorno que indica se o processo terminou com erro ou não.
- O código de retorno do último comando executado fica armazenado na variável especial $?
- `0` indica sucesso.
- Qualquer retorno diferente de `0` indica falha.

# Exit status

```
$ ls ~
Área de Trabalho  Downloads  lista_sites  Público             Vídeos
chamada.txt       Dropbox    Modelos      realoc_uids.py.old  vpn_lncc
Documentos        Imagens    Música       repos
$ echo $?
0

$ ls nao_existe
ls: impossível acessar nao_existe: Arquivo ou diretório não encontrado
$ echo $?
2
```

# Exit status

| Código | Significado |
|---|---|
| 0 | Execução terminou sem erros |
| 1 | Maioria dos erros comuns |
| 2 | Erro de uso de 'builtin' do shell |
| 126 | Sem permissão de execução |
| 127 | Comando não encontrado |

Para ter certeza dos códigos de retorno dos comandos, leia a manpage com o comando man comando.

# Condicional (`if`)

- Ao contrário de outras linguagens, que testam expressões, o if do shell testa o sucesso de execução de um comando, através de seu código de retorno.
- Sintaxe:

```
if COMANDO
then
   comandos caso sucesso
else
   comandos caso falha
fi
```

# Condicional (if)

- Sintaxe 2, encadeando ifs:

```
if COMANDO
then
   comandos caso sucesso de COMANDO
elif COMANDO2
   comandos caso sucesso de COMANDO2
else
   comandos caso falha
fi
```

# Condicional (if)

- Exemplo: script que testa a conexão com a internet.

```
1 #!/bin/bash
2 SITE="google.com"
3
4 if ping -c 5 "$SITE" &>/dev/null
5 then
6  echo "Conexão com a internet funcionando"
7 else
8  echo "Sem conexão com a internet"
9 fi
```

# Condicional (`if`)

- O operador lógico `&&` só executa o comando seguinte caso o primeiro tenha código de retorno 0.
- `comando && comando caso sucesso`
- O operador lógico `||` é o inverso.

# Condicional (`if`)

- Exemplo: script que testa a conexão com a internet (com operadores && e ||).

```
1 #!/bin/bash
2 SITE="google.com"
3
4 ping -c 5 $SITE &>/dev/null && echo "Conexão com a internet funcionando"
  || echo "Sem conexão com a internet"
```

# Testando expressões (`test`)

- Para testar expressões, como em outras linguagens de programação, use o `if` junto com o comando `test`.
- O comando `test` testa uma expressão e retorna sucesso (0) caso a expressão seja verdadeira e falha (1) caso seja falsa.
- `test EXPRESSAO`

# Testando expressões (test)

- Exemplo: testando se um arquivo existe.

```
$ test -e /etc/passwd
$ echo $?
0

$ test -e nao_existe
$ echo $?
1
```

# Testando expressões (test)

- Exemplo 2: testando inteiros.

```
$ valor=10
$ test "$valor" -gt 5
$ echo $?
0

$ test "$valor" -lt 5
$ echo $?
1
```

# Testando expressões (test)

- Exemplo 2: testando strings.

```
$ msg="olá mundo"
$ test "$msg" = "olá mundo"
$ echo $?
0

$ test "$msg" != "hello world"
$ echo $?
0
```

# Testando expressões (test)

- Exemplo 3: `if` + `test`.

```
1  #!/bin/bash
2
3  echo "Posso buscar dados do sistema? [sn]"
4  read RESPOSTA
5
6  if test $RESPOSTA = 'n' -o $RESPOSTA = 'N'
7  then
8          exit 1
9  else
10         echo "Usuários conectados:"
11         who
12         echo -e "\nUso de disco:"
13         df
14 fi
```

# Testando expressões ([ ])

- Na maioria dos shells o comando `test` pode ser substituído pelo comando `[`, terminado pelo argumento `]`.
- Isso deixa o `if` com sintaxe bem semelhante à das outras linguagens.
- `[ EXPRESSAO ]`
- Não esqueça dos espaços em branco entre os colchetes e a expressão.

# Testando expressões ([ ])

- Exemplo: sintaxe alternativa do `test`.

```
1  #!/bin/bash
2
3  echo "Posso buscar dados do sistema? [sn]"
4  read RESPOSTA
5
6  if [ $RESPOSTA = 'n' -o $RESPOSTA = 'N' ]
7  then
8          exit 1
9  else
10         echo "Usuários conectados:"
11         who
12         echo -e "\nUso de disco:"
13         df
14 fi
```

# Testando expressões ([[ ]])

- O bash e o ksh têm um builtin `[[`, que também testa expressões.
- Possui algumas diferenças do `test` clássico.
- Se o script precisar ser portável, evite o uso.
- http://mywiki.wooledge.org/BashFAQ/031

# Testando expressões ([[ ]])

- Exemplo: builtin [[, alternativa ao `test` clássico.

```
1  #!/bin/bash
2
3  echo "Posso buscar dados do sistema? [sn]"
4  read RESPOSTA
5
6  if [[ $RESPOSTA = 'n' || $RESPOSTA = 'N' ]]
7  then
8          exit 1
9  else
10         echo "Usuários conectados:"
11         who
12         echo -e "\nUso de disco:"
13         df
14 fi
```

# Opções do test ou [

| Comparação numérica | |
|---|---|
| -lt | É menor que (LessThan) |
| -gt | É maior que (GreaterThan) |
| -le | É menor igual (LessEqual) |
| -ge | É maior igual (GreaterEqual) |
| -eq | É igual (EQual) |
| -ne | É diferente (NotEqual) |

Fonte: http://aurelio.net/shell/canivete/

# Opções do test ou [

| Testes em arquivos | |
|---|---|
| -b | É um dispositivo de bloco |
| -c | É um dispositivo de caractere |
| -d | É um diretório |
| -e | O arquivo existe |
| -f | É um arquivo normal |
| -g | O bit SGID está ativado |
| -G | O grupo do arquivo é o do usuário atual |
| -k | O sticky-bit está ativado |
| -L | O arquivo é um link simbólico |
| -O | O dono do arquivo é o usuário atual |

# Opções do test ou [

| Testes em arquivos (2) | |
|---|---|
| -p | O arquivo é um named pipe |
| -r | O arquivo tem permissão de leitura |
| -s | O tamanho do arquivo é maior que zero |
| -S | O arquivo é um socket |
| -t | O descritor de arquivos N é um terminal |
| -u | O bit SUID está ativado |
| -w | O arquivo tem permissão de escrita |
| -x | O arquivo tem permissão de execução |
| -nt | O arquivo é mais recente (NewerThan) |
| -ot | O arquivo é mais antigo (OlderThan) |
| -ef | O arquivo é o mesmo (EqualFile) |

# Opções do test ou [

| Comparação de cadeias de caracteres | |
|---|---|
| = | É igual |
| != | É diferente |
| -n | É não nula |
| -z | É nula |

| Operadores lógicos | |
|---|---|
| ! | Não lógico |
| -a | E lógico |
| -o | Ou lógico |

# Repetição (`while`)

- O `while` do shell também testa o sucesso da execução de um comando, ao invés de uma expressão.
- Sintaxe:

```
while COMANDO
do
   comandos caso COMANDO retorne sucesso
done
```

# Repetição (while)

- Exemplo: contar de 0 a 10.

```
1 #!/bin/bash
2
3 x=0
4 while [ "$x" -le 10 ]
5 do
6     echo $x
7     x=$((x+1))
8 done
```

# Repetição (`while`)

- Um uso típico do `while` é iterar sobre as linhas de um arquivo.
- Existem várias formas de se fazer isso, veremos algumas.
- O comando `read` lê uma linha da entrada padrão.
- http://mywiki.wooledge.org/BashFAQ/001

# Repetição (while)

- Exemplo: lista arquivo /etc/passwd, numerando as linhas.

```
1 #!/bin/bash
2
3 X=1
4 while read LINHA;
5 do
6    echo "$X $LINHA"
7    X=$((X+1))
8 done < /etc/passwd
```

# Repetição (while)

- Exemplo 2: lista arquivo /etc/passwd, numerando as linhas.

```
1 #!/bin/bash
2
3 X=1
4 cat /etc/passwd | while read LINHA;
5 do
6    echo "$X $LINHA"
7    X=$((X+1))
8 done
```

# Repetição (until)

- O comando `until` é análogo ao `while`, só que a repetição para quando o `COMANDO` falha.
- Sintaxe:

```
until COMANDO
do
   comandos caso COMANDO retorne falha
done
```

# Repetição (until)

- Exemplo: contar de 0 a 10.

```
1 #!/bin/bash
2
3 x=0
4 until [ "$x" -gt 10 ]
5 do
6     echo $x
7     x=$((x+1))
8 done
```

# Repetição (for)

- O comando `for` percorre uma lista e a atribui a uma variável cada valor dessa lista.
- Sintaxe:

```
for ELEMENTO in VALOR1 VALOR2 VALOR3
do
   comandos que manipulam ELEMENTO
done
```

# Repetição (for)

- Exemplo: contar de 0 a 10.
- Perceba que os itens da lista são separados por espaços em branco.

```
1 #!/bin/bash
2
3 for X in 0 1 2 3 4 5 6 7 8 9 10
4 do
5     echo $X
6 done
```

# Repetição (for)

- Exemplo: contar de 0 a 10.
- O comando `seq A B`, escreve de `A` a `B` na saída padrão.

```
1 #!/bin/bash
2
3 for X in $(seq 0 10)
4 do
5     echo $X
6 done
```

# Repetição (for)

- Também é possível utilizar uma sintaxe parecida com o `for` da linguagem C no shell.
- Sintaxe:

```
for ((INICIALIZACAO, TESTE, INCREMENTO))
do
   comandos que manipulam ELEMENTO
done
```

# Repetição (for)

- Exemplo: contar de 0 a 10.

```
1 #!/bin/bash
2
3 for ((X=0;X<=10;X++))
4 do
5     echo $X
6 done
```

# Controle de fluxo (case)

- O comando `case` permite executar comandos baseados em correspondência de padrões.
- Pode substituir vários `if`s aninhados.
- Sintaxe:

```
case VARIAVEL in
   txt1) comandos ;;
   txt2) comandos ;;
   txt3) comandos ;;
   *) comandos ;;
esac
```

# Controle de fluxo (case)

```
 1 #!/bin/bash
 2
 3 echo "Que dados do sistema deseja exibir?
 4 1) usuários conectados
 5 2) uso de disco
 6 3) uso de memória
 7 4) data e hora
 8
 9 Entre qualquer outra opção para sair"
10 read RESPOSTA
11
12 case "$RESPOSTA" in
13         1) echo "Usuários conectados:"; who ;;
14         2) echo "Uso de disco:"; df ;;
15         3) echo "Uso de memória:"; free -m ;;
16         4) echo "Data e hora:"; date ;;
17         *) exit 1 ;;
18 esac
```

# Coringas para o case

| Coringa | Casa com |
|---|---|
| * | Qualquer coisa |
| ? | Um caractere qualquer |
| [...] | Qualquer um dos caracteres listados |
| [^...] | Qualquer caractere, exceto os listados |
| ...\|... | Qualquer dos textos divididos por \| |

Fonte: http://aurelio.net/shell/canivete/

# Controle de fluxo (case)

```
 1  !/bin/bash
 2
 3 echo "Que dados do sistema deseja exibir?
 4 1) (u)suários conectados
 5 2) uso de (d)isco
 6 3) uso de (m)emória
 7 4) data e (h)ora"
 8 read RESPOSTA
 9
10 case "$RESPOSTA" in
11         [uU]) echo "Usuários conectados:"; who ;;
12         [dD]) echo "Uso de disco:"; df ;;
13         [mM]) echo "Uso de memória:"; free -m ;;
14         [Hh]) echo "Data e hora:"; date ;;
15         [^uUdDmMhH]) echo "Opção desconhecida"; exit 1;;
16 esac
```

# Funções

- Assim como em outras linguagens é possível definir funções em shell script.
- Sintaxe:

```
foo(){
   comando1
   comando2
   return EXIT_CODE
}
```

- Se um `EXIT_CODE` não for fornecido, é retornado o do último comando.

# Passagem de parâmetros

- Os parâmetros podem ser acessados pelas variáveis `$0` até `$9`.
- O parâmetro `$0` é o nome da função.
- Do décimo parâmetro em diante, usar `${10}`.
- Mais de 10 parâmetros só funciona no bash e no ksh; no bourne shell, não.
- A variável `$@` guarda todos os parâmetros. Usar sempre entre aspas duplas.

# Passagem de parâmetros

- Exemplo: listar argumentos passados para função.

```
 1 #!/bin/bash
 2
 3 foo(){
 4     x=1
 5     for arg in "$@"
 6     do
 7         echo "$xº parâmetro: $arg"
 8         x=$((x+1))
 9     done
10 }
11
12 foo 1 '2 3' 4 5 'olá mundo'
```

# Passagem de parâmetros

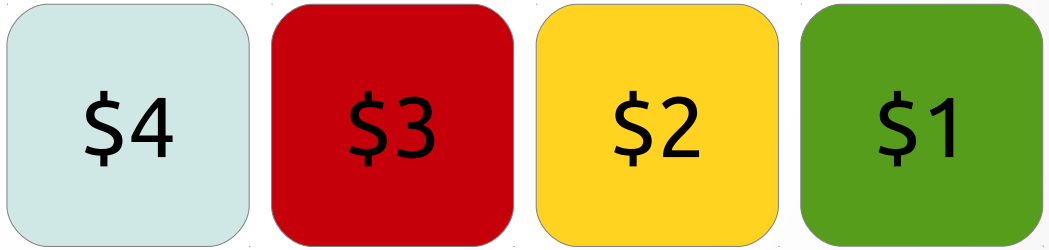
- Executando o script...

```
$ ./exemplo_funcao.sh
1º parâmetro: 1
2º parâmetro: 2 3
3º parâmetro: 4
4º parâmetro: 5
5º parâmetro: olá mundo
```

# Comando shift

Antes do `shift`


Depois do `shift`



# Comando shift

- Exemplo: listar argumentos passados para função.

```
 1 #!/bin/bash
 2
 3 foo(){
 4     x=1
 5     while [ -n "$1" ]
 6     do
 7         echo "$xº parâmetro: $1"
 8         x=$((x+1))
 9         shift
10     done
11 }
12
13 foo 1 '2 3' 4 5 'olá mundo'
```

# Expressões aritméticas

- O bash suporta apenas aritmética de inteiros, via a contrução `$((...))`
- Para fazer aritmética de ponto flutuante é necessário um programa externo, como o `bc`.
- Alguns shells como o ksh e csh suportam aritmética de ponto flutuante nativamente.

# Expressões aritméticas

- Exemplo: fazer 5/2 e 10/3

```
$ echo $((5/2))
2
$ echo $((10/3))
3

$ echo "scale=3; 5/2" | bc
2.500
$ echo "scale=3; 10/3" | bc
3.333
```

# Expressões aritméticas

- Exemplo: fazer 10/3, usando variáveis

```
$ DIVIDENDO=10
$ DIVISOR=3
$ RESULT=$((DIVIDENDO/DIVISOR))
$ echo $RESULT
3

$ RESULT=$(echo "scale=3; $DIVIDENDO/$DIVISOR" | bc)
$ echo $RESULT
3.333
```