Tópicos Especiais em Fundamentos da Computação
Shell Scripting

Estruturas de Fluxo Condicionais e de Repetição

Andrei Rimsa Álvares
andrei@cefetmg.br

CEFET-MG

DECOM

Sumário

- Introdução
- Status de saída
- Comando `test`
- Estruturas de fluxo condicionais
  - Comandos `if` e `case`
  - Outras formas de desvios
- Estruturas de repetição
  - Comandos `for`, `while` e `until`
  - Sequenciadores `break` e `continue`
  - Comando select

INTRODUÇÃO

Controle de Fluxo

- A *shell* usa as mesmas construções para controle de fluxo que linguagens de programação de alto nível
  - Controle de fluxo condicional
    - `if` e `case`
  - Controle de fluxo de repetição
    - `for`, `while`, `until` e `select`
  - Sequenciadores
    - `break` e `continue`

Exemplo (Condicional)

- Exemplo de desvio condicional, onde **se** x = 5 imprime “X igual a 5”, **senão** imprime “X não é igual a 5”

check_five

```
#!/bin/bash

x=$1

if [ $x = 5 ]; then
  echo "x é igual a 5"
else
  echo "x não é igual a 5"
fi
```

```
$ ./check_five 5
x é igual a 5
$ ./check_five 7
x não é igual a 5
```

Mesmo Exemplo (Condicional)

- Mesmo exemplo, mas executando pela linha de comando

```
$ x=5
$ if [ $x = 5 ]; then echo "x é igual a 5”;
> else echo "x não é igual a 5"; fi
x é igual a 5
$ x=7
$ if [ $x = 5 ]; then echo "x é igual a 5”;
> else echo "x não é igual a 5"; fi
x não é igual a 5
$
```

Dica: pode usar o comando !-2 para repetir o penúltimo comando

Exemplo (Repetição)

- Exemplo de fluxo de repetição, onde são exibidos número sequencialmente de 1 a 5

count_5

```
#!/bin/bash

count=1
while [ $count -le 5 ]; do
  echo $count;
  count=$((count + 1))
done
```

```
$ ./count_5
1
2
3
4
5
```

STATUS DE SAÍDA

Status de Saída

- **Relembrando:** todo comando retorna para a *shell* um valor (de 0 a 255) indicando sucesso (0) ou falha em sua execução (diferente de 0)

```
$ ls -d /usr/bin
/usr/bin
$ echo $?
0
$ ls -d /bin/usr
ls: /bin/usr: No such file or directory
$ echo $?
2
$
```

`true` e `false`

- Existem dois comandos internos da shell extremamente simples que apenas terminam com status 0 ou 1
  - o comando `true` sempre executa com sucesso (retorna 0)
  - o comando `false` sempre executa sem sucesso (retorna 1)

```
$ true
$ echo $?
0
$ false
$ echo $?
1
$
```

Esses dois comandos serão utilizados para demonstrar o funcionamento do comando **if**

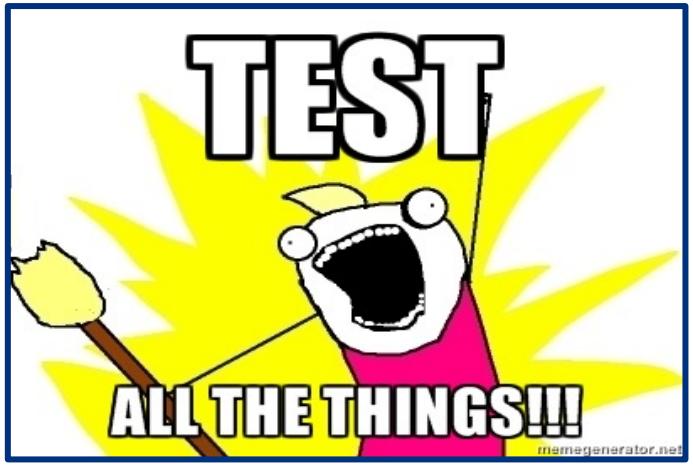
Comando `if`

- O que o comando `if` faz é avaliar o comando especificado verificando seu status de saída

```
$ if true; then echo "É verdade"; fi
É verdade
$ if false; then echo "É verdade"; fi
$
```

O que acontece se for executado mais de um comando (separados por ponto-vírgula)?

COMANDO `test`



Controle de Fluxo

- De longe, o comando mais frequentemente usado com comandos condicionais é **`test;`** faz uma variedade de verificações/comparações

- Possuem duas formas equivalentes, onde expression é avaliado em verdadeiro ou falso, retornando 0 quando a expressão é verdadeira e 1 quando é falsa

1) **`test expression`**

```
Ex.: $ test $x = 5
     $ echo $?
     0
```

2) **`[ expression ]`**

```
Ex.: $ [ $x = 5 ]
     $ echo $?
     0
```

Controle de Fluxo

- O comando test é capaz de avaliar

  1) Expressões com arquivos
  2) Expressões com strings
  3) Expressões com inteiros

Expressões com Arquivos

- Tabela com as expressões com arquivos

Expressão	É verdade se
`file1 –ef file2`	`file1` e `file2` possuem o mesmo número de inode (hardlink)
`file1 –nt file2`	`file1` é mais novo que `file2`
`file1 –ot file2`	`file1` é mais velho que `file2`
`-b file`	arquivo existe e é um arquivo (dispositivo) de bloco-especial
`-c file`	arquivo existe e é um arquivo (dispositivo) de caractere-especial
`-d file`	arquivo existe e é um diretório
`-e file`	arquivo existe
`-f file`	arquivo existe e é um arquivo regular
`-g file`	arquivo existe e é *setgid*
`-G file`	arquivo existe e pertence ao ID do grupo efetivo
`-k file`	arquivo existe e tem o *stick bit* definido

Expressões com Arquivos

- ... continuação da tabela de expressões com arquivos

Expressão	É verdade se
`-L file`	arquivo existe e é um *link* simbólico
`-O file`	arquivo existe e pertence ao ID do usuário efetivo
`-p file`	arquivo existe e é um *pipe* nomeado
`-r file`	arquivo existe e é legível (pelo usuário efetivo)
`-s file`	arquivo existe e tem tamanho maior que zero (não vazio)
`-S file`	arquivo existe e *socket* de rede
`-t fd`	`fd` é um descritor de arquivos redirecionado de/para o terminal
`-u file`	arquivo existe e é *setuid*
`-w file`	arquivo existe e gravável (pelo usuário efetivo)
`-x file`	arquivo existe e é executável (pelo usuário efetivo)

Exemplo

- Exemplos de expressões com arquivos

Dica: envolva o arquivo entre aspas

```
$ FILE=~/.bashrc
$ [ -e "$FILE" ] # arquivo existe?
$ echo $?
0
$ [ -f "$FILE" ] # arquivo existe e é regular?
$ echo $?
0
$ [ -d "$FILE" ] # arquivo existe e é diretório?
$ echo $?
1
$ [ -w "$FILE" ] # arquivo existe e é gravável?
$ echo $?
0
$ [ -x "$FILE" ] # arquivo existe e é executável?
$ echo $?
1
```

Expressões com Strings

- Tabela com as expressões com strings

Expressão	É verdade se
`string`	a `string` não é *null* (definida ou não vazia)
`-n string`	o tamanho da `string` é maior que zero
`-z string`	o tamanho da `string` é zero
`string1 = string2` `string1 == string2`	`string1` e `string2` são iguais (símbolo de igual único ou duplo pode ser usado, mas com dois iguais é preferencial)
`string1 != string2`	`string1` e `string2` não são iguais (diferentes)
`string1 > string2`	`string1` vem depois de `string2` se ordenadas
`string1 < string2`	`string1` vem antes de `string2` se ordenadas

Exemplo

- Exemplos de expressões com strings

Dica: envolva a string entre aspas

Cuidado: envolva sempre os símbolos de < e > entre aspas

```
$ ANSWER=maybe
$ [ -z "$ANSWER" ] # Tem tamanho zero?
$ echo $?
1
$ [ -n "$ANSWER" ] # Tem algum tamanho?
$ echo $?
0
$ [ "$ANSWER" == "yes" ] # É igual a yes?
$ echo $?
1
$ [ "$ANSWER" == "maybe" ] # É igual a maybe?
$ echo $?
0
$ [ "$ANSWER" "<" "no" ] # Vem antes de no?
$ echo $?
0
```

Expressões com Inteiros

- Tabela com as expressões com inteiros

Expressão	É verdade se
`integer1 -eq integer2`	`integer1` é igual a `integer2`
`integer1 -ne integer2`	`integer1` é diferente de `integer2`
`integer1 -le integer2`	`integer1` é menor ou igual a `integer2`
`integer1 -lt integer2`	`integer1` é menor que `integer2`
`integer1 -ge integer2`	`integer1` é maior ou igual a `integer2`
`integer1 -gt integer2`	`integer1` é maior que `integer2`

Exemplo

- Exemplos de expressões com inteiros

Dica: envolva o inteiro entre aspas

Cuidado: funciona somente com números inteiros

```
$ INT=-5
$ [ "$INT" -eq 0 ] # É igual a zero?
$ echo $?
1
$ [ "$INT" -lt 0 ] # É negativo?
$ echo $?
0
$ [ "$((INT % 2))" -eq 0 ] # É par?
$ echo $?
1
```

Comando [[ ]]

- Versões recentes do bash adicionam um comando composto que age como um aprimoramento do comando **`test`**

  `[[ expression ]]`

  - onde `expression` é uma expressão que avalia verdadeiro ou falso

- O comando `[[ ]]` é muito similar ao comando test (suporta todas as suas expressões), mas adiciona uma nova comparação de strings

  `string1 =~ regex`

  - que retorna verdadeiro se `string1` casa com a expressão regular estendida `regex`

Permite validações de dados mais poderosas!

Exemplo

- Se a variável não fosse inteira, a comparação inteira com zero poderia gerar um erro

```
$ INT=abc
$ [ "$INT" -eq 0 ] # É igual a zero?
-bash: [: abc: integer expression expected
$ echo $?
2
```

- Mas poderia-se tentar validar o número antes de realizar tal operação

```
$ INT=-5
$ [[ "$INT" =~ ^-?[0-9]+$ ]] # É número inteiro?
$ echo $?
0
```

Outro Exemplo

- Outra funcionalidade do comando [[ ]] é que o operador == suporta o casamento de padrões da mesma forma que expansão de arquivos

```
$ FILE=foo.bar
$ [[ "$FILE" == foo.* ]]
$ echo $?
0
```

Comando (( ))

- O bash também proporciona o comando composto (( )) que é útil para realizar operações inteiras, suporta o conjunto completo para avaliações aritméticas

- Esse comando é útil para fazer testes verdade aritméticas, resultando em verdadeiro se a avaliação aritmética for não-zero

```
$ ((1)) # É verdade?
$ echo $?
0
$ ((0)) # É falso?
$ echo $?
1
```

Combinando Expressões

- Também é possível combinar expressões para criar avaliações mais complexas; usando operadores lógicos

- Existem três operadores lógicos (**AND**, **OR** e **NOT**) para os comandos `test`, `[[ ]]` e `(( ))`; onde têm sintaxe diferenciada

Operação	`test` e `[ ]`	`[[ ]]` e `(( ))`
AND	`-a`	`&&`
OR	`-o`	`||`
NOT	`!`	`!`

Exemplo

- Combinando expressões com **test**, [ ] e [[ ]]

```
$ INT=50
$ test "$INT" -ge 1 -a "$INT" -le 100
$ echo $?
0
$ [ "$INT" -ge 1 -a "$INT" -le 100 ]
$ echo $?
0
$ [[ "$INT" -ge 1 && "$INT" -le 100 ]]
$ echo $?
0
```

Outro Exemplo

- Para negar uma combinação de expressões, deve-se usar parênteses envolvendo as expressões

  - Usando o comando [[ ]]

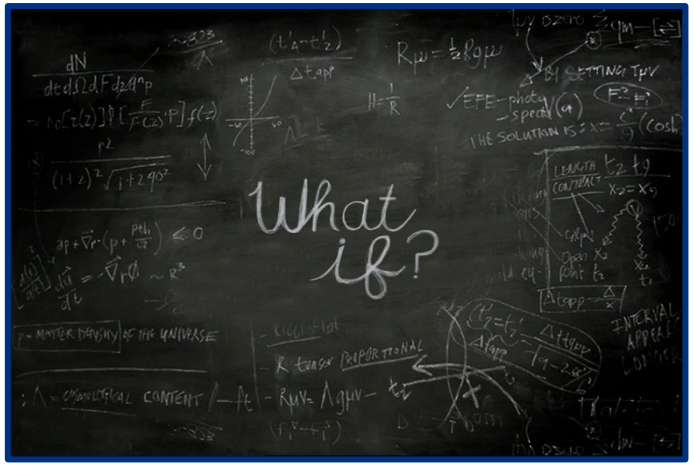
```
$ INT=50
$ [[ ! ("$INT" -ge 1 && "$INT" -le 100) ]]
$ echo $?
1
```

  - Usando o comando [ ]

```
$ INT=50
$ [ ! \( "$INT" -ge 1 -a "$INT" -le 100 \) ]
$ echo $?
1
```

Nesse caso é preciso escapar os parênteses

COMANDO `if`



Comando `if`

- O comando `if` tem a seguinte sintaxe

```
if command; then
    commands
[elif commands; then
    commands...]
[else
    commands]
fi
```

As partes entre colchetes são **opcionais**

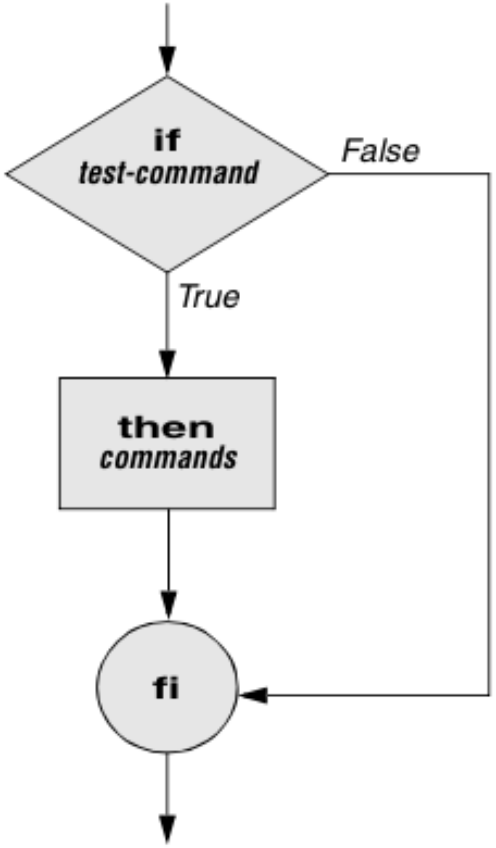
Dica: o ponto-vírgula pode ser suprimido se **`then`** vier na próxima linha

- Para exemplificar, o comando `if` será quebrado em três partes
  - `if...then`
  - `if...then...else`
  - `if...then...elif`

`if...then`

- A estrutura de controle **`if...then`** possui a seguinte sintaxe

```
if test-command; then
    commands
fi
```



Exemplo

- Verificar se duas palavras são iguais

Usando o comando **test** como condição de teste

if1

```
#!/bin/bash

echo -n "word 1: "
read word1

echo -n "word 2: "
read word2

if test "$word1" = "$word2"; then
    echo "Match"
fi

echo "End of program"
```

```
$ ./if1.sh
word 1: apple
word 2: apple
Match
End of program
$ ./if1.sh
word 1: apple
word 2: peach
End of program
$
```

Outro Exemplo

- Verifica se o programa recebe um argumento

chkargs

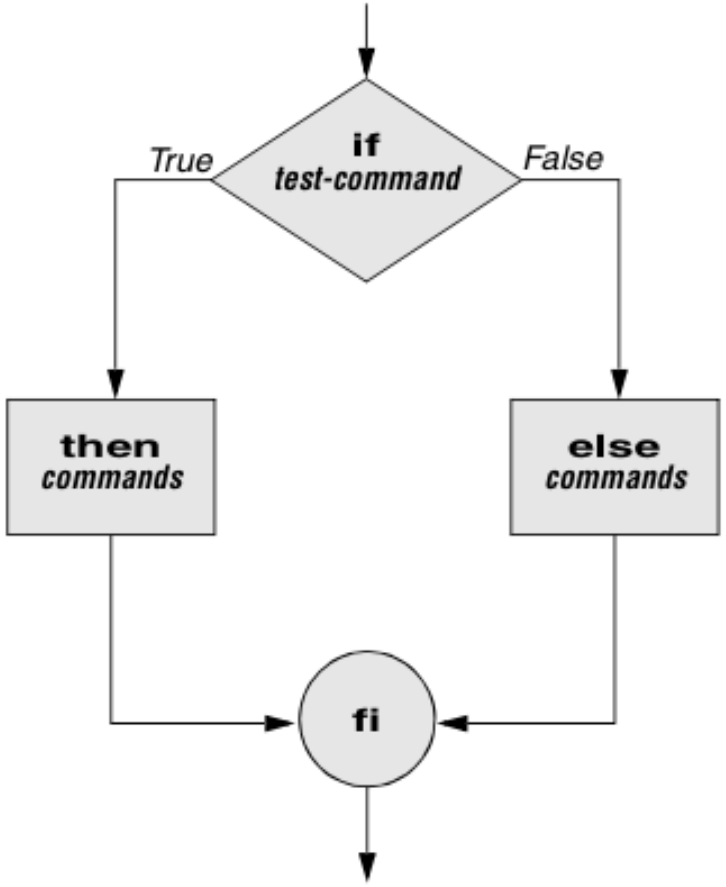
```
#!/bin/bash

if [ $# -eq 0 ]; then
    echo "Usar: $0 [Argumento]" 1>&2
    exit 1
fi

echo "Executando com '$1'"
```

```
$ ./chkargs
Usar: ./chkargs [Argumento]
$ ./chkargs abc
Executando com 'abc'
$
```

`if...then...else`

- O comando **`else`** adiciona um desvio de dois caminhos ao **`if`**

```
if test-command; then
    commands
else
    commands
fi
```



Exemplo

- Recebe um arquivo como parâmetro e verifica se é arquivo regular

is_ordfile

```
#!/bin/bash

if [ $# -eq 0 ]; then
    echo "Usar: $0 [Arquivo]" 1>&2
    exit 1
fi

if [ -f "$1" ]; then
    echo "$1: é arquivo regular"
else
    echo "$1: não é arquivo regular"
fi
```

```
$ ./is_ordfile
Usar: ./is_ordfile [Arquivo]
$ ./chkargs /etc/passwd
/etc/passwd: é arquivo regular
$ ./chkargs /etc
/etc: não é arquivo regular
```

Outro Exemplo

- Ler vários arquivos recebidos via parâmetros; se usar **–v**, ler com **less**, caso contrário com **cat**

readfiles

```
#!/bin/bash

if [ $# -eq 0 ]; then
    echo "Usage: $0 <-v> [filenames...]" 1>&2
    exit 1
fi

if [ "$1" = "-v" ]; then
    shift
    less -- "$@"
else
    cat -- "$@"
fi
```

```
$ ./readfiles /etc/passwd
...
$ ./readfiles –v /etc/passwd
...
```

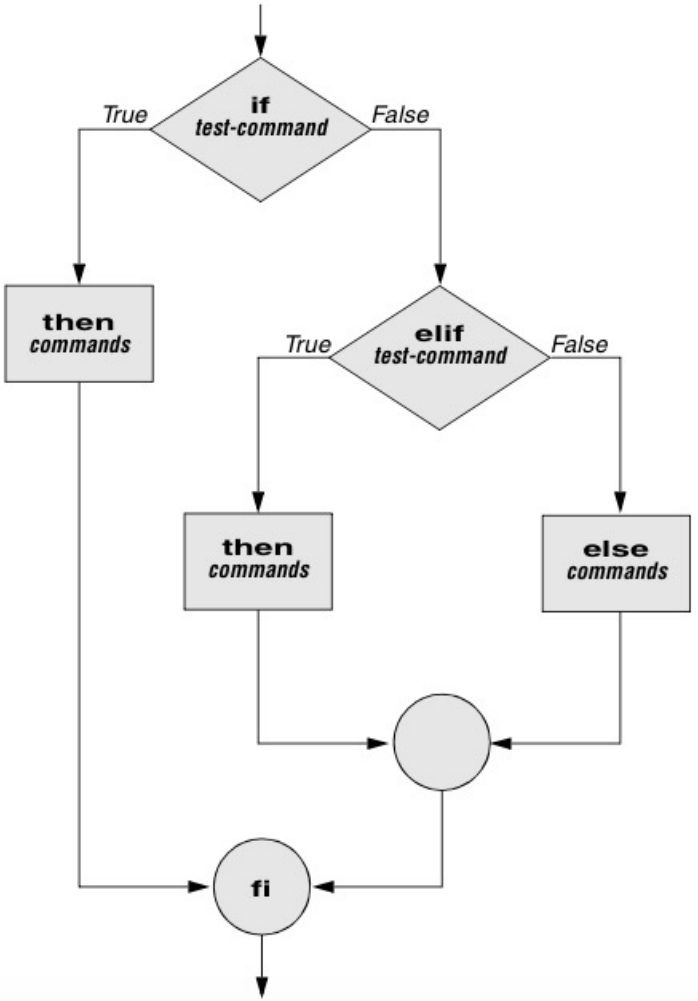
O comando **shift** foi usado para deslocar os parâmetros após -v

O parâmetro -- foi usado para evitar a injeção de opcionais para os comandos **less** ou **cat**

`if...then...elif`

- O comando **`elif`** combina o comando **`if`** com **`else`** permitindo construir desvios aninhados

```
if test-command; then
    commands
elif test-command; then
    commands
...
else
    commands
fi
```



Exemplo

- Verificar quais, entre três palavras, são semelhantes

if3

```
#!/bin/bash

echo -n "word 1: "
read word1
echo -n "word 2: "
read word2
echo -n "word 3: "
read word3

if [ "$word1" = "$word2" -a "$word2" = "$word3" ];
then
    echo "Match: words 1, 2 & 3"
elif [ "$word1" = "$word2" ] then
    echo "Match: words 1 & 2"
elif [ "$word1" = "$word3" ] then
    echo "Match: words 1 & 3"
elif [ "$word2" = "$word3" ] then
    echo "Match: words 2 & 3"
else
    echo "No match"
fi
```

```
$ ./if3
word 1: apple
word 2: orange
word 3: pear
No match
$ ./if3
word 1: apple
word 2: orange
word 3: apple
Match: words 1 & 3
$ ./if3
word 1: apple
word 2: apple
word 3: apple
Match: words 1, 2 & 3
```

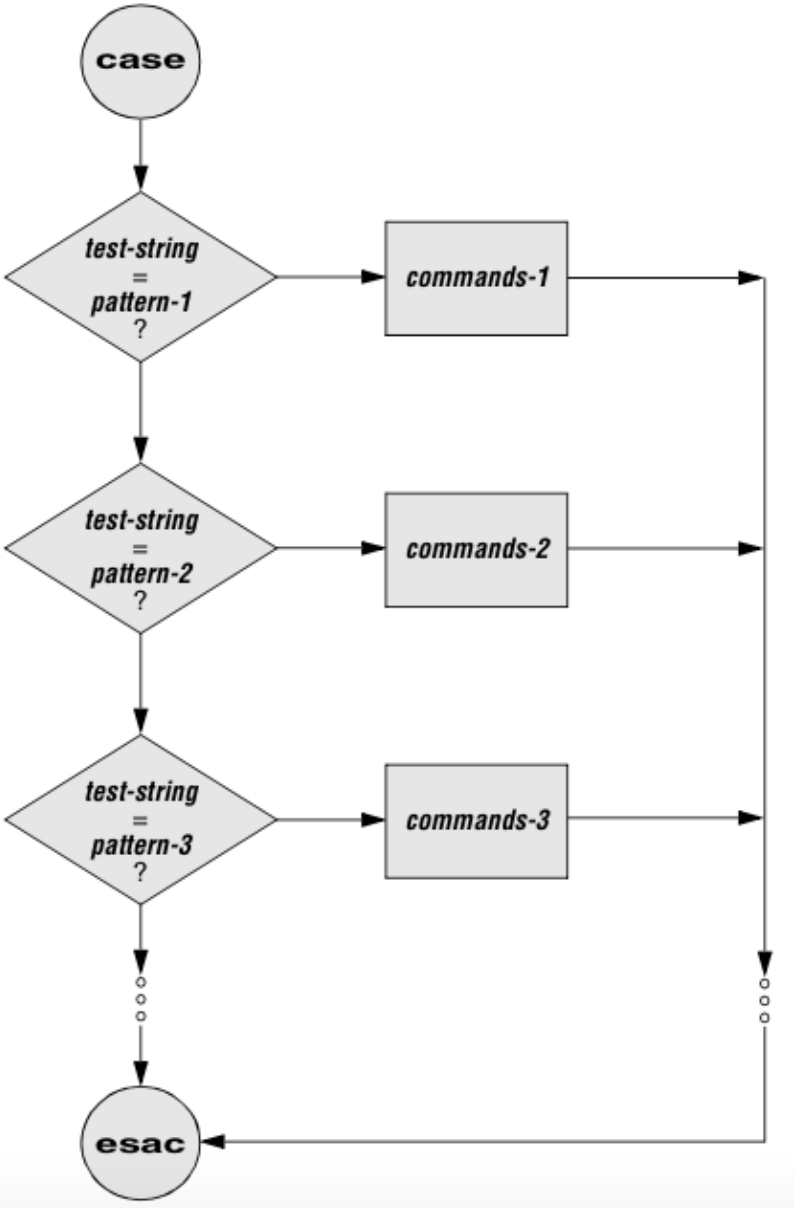
COMANDO case

Comando `case`

- A estrutura de fluxo **`case`** é um mecanismo de desvio com múltiplos desvios

```
case test-string in
    pattern-1)
        commands-1
        ;;
    pattern-2)
        commands-2
        ;;
    pattern-3)
        commands-3
        ;;
    ...
esac
```

Similar ao comando switch de LP's



Exemplo

- Verificar se o usuário selecionou as opções A, B ou C; caso contrário imprimir mensagem de erro

case1

```
#!/bin/bash

echo -n "Enter A, B, or C: "
read letter

case "$letter" in
    A)
        echo "You entered A"
        ;;
    B)
        echo "You entered B"
        ;;
    C)
        echo "You entered C"
        ;;
    *)
        echo "You did not enter A, B or C"
        ;;
esac
```

```
$ ./case1
Enter A, B, or C: B
You entered B
$ ./case1
Enter A, B, or C: b
You did not enter A, B or C
$
```

O asterisco (*) indica qualquer string de caracteres

Dica: use no final para pegar qualquer padrão (caso padrão)

Padrões Especiais

- O padrão em uma estrutura case é análogo a uma referência a arquivos ambíguos

Padrão	Função
*	Casa qualquer sequência de caracteres (use como caso padrão)
?	Casa qualquer caractere único
[...]	Define uma classe de caracteres, onde cada caractere envolto nos colchetes são testados um por vez (aceita hífen para especificar uma faixa de caracteres)
|	Separa escolhas alternativas que satisfazem determinado caso

Exemplo

- Reescrevendo o exemplo anterior, mas *case insensitive*

case2

```
#!/bin/bash

echo -n "Enter A, B, or C: "
read letter

case "$letter" in
    a|A)
        echo "You entered A"
        ;;
    b|B)
        echo "You entered B"
        ;;
    c|C)
        echo "You entered C"
        ;;
    *)
        echo "You did not enter A, B or C"
        ;;
esac
```

```
$ ./case1
Enter A, B, or C: B
You entered B
$ ./case1
Enter A, B, or C: b
You entered B
$
```

Menu

- O comando case é muito útil para fazer menus

cmdmenu

```
#!/bin/bash

echo "COMMAND MENU"
echo
echo "  a. Current date and time"
echo "  b. Users currently logged in"
echo "  c. Name of the working directory"
echo "  d. Contents of the working directory"
echo
echo -n "Enter a, b, c or d: "
read answer
echo

case "$answer" in
    a)
        date
        ;;
    b)
        who
        ;;
    c)
        pwd
        ;;
    d)
        ls
        ;;
    *)
        echo "Invalid selection: $answer"
        ;;
esac
```

```
$ ./cmdmenu
COMMAND MENU

  a. Current date and time
  b. Users currently logged in
  c. Name of the working directory
  d. Contents of the working directory

Enter a, b, c or d: a

Sun Nov 30 14:44:08 BRST 2014
$
```

Existe uma forma melhor de fazer menus?

OUTRAS FORMAS DE DESVIOS

Outras Formas de Desvios

- Bash provê dois operadores que podem realizar desvios

1) **&&** (AND): o **comando2** só é executado se o **comando1** executar com sucesso (status de saída igual a zero)

comando1 && comando2

2) **||** (OR): o **comando2** só é executado se o **comando1** executar sem sucesso (status de saída diferente de zero)

comando1 || comando2

Exemplo

- Exemplo 1: irá criar o diretório *temp*, e se tiver êxito, então mudar o diretório atual para *temp*

```
$ mkdir temp && cd temp
```

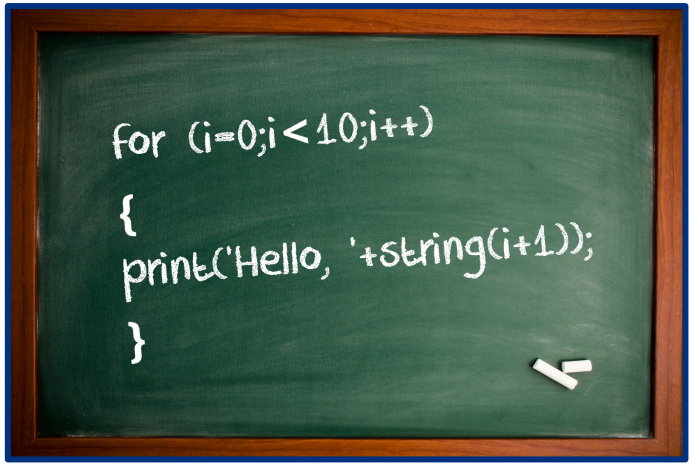
- Exemplo 2: verificar se *temp* existe e é um diretório e se não for o caso, criar o diretório com esse nome

```
$ [ -d temp ] || mkdir temp
```

- Exemplo 3: abortar o script se temp não existir e se não for um diretório (construção muito útil na prática)

```
$ [ -d temp ] || exit 1
```

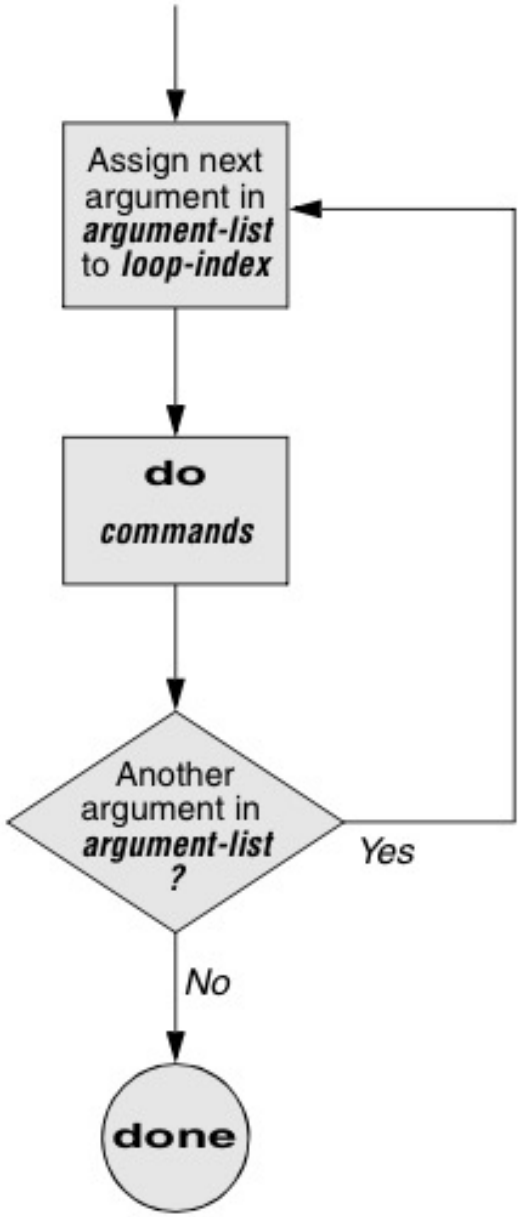
COMANDO `for`



for...in

- A estrutura de controle de repetição **for...in** possui a seguinte sintaxe

```
for loop-index in argument-list; do
    commands
done
```



Exemplo

- Exemplo onde cada fruta (*apples*, *oranges*, *pears* e *bananas*) é associada a variável `fruit`, que posteriormente é impressa na tela, até esgotar a lista por completo

fruit

```
#!/bin/bash

for fruit in apples oranges pears bananas; do
    echo "$fruit"
done
echo "Task complete"
```

```
$ ./fruit
apples
oranges
pears
bananas
Task complete
```

Expansões na Lista de Argumentos

- A lista de argumentos do comando **`for`** aceita expansões de arquivo

- Exemplo: listar somente os diretórios da pasta atual de trabalho

dirfiles

```
#!/bin/bash

for  i in *; do
        if [ -d "$i" ]; then
                echo "$i"
        fi
done
```

```
$ ./dirfiles
Applications
Desktop
Documents
Downloads
Library
Movies
Music
Pictures
Public
```

Outro Exemplo

- Usando comandos como expansão na lista de parâmetros para listar os nomes dos identificadores dos grupos do usuário

groups

```
#!/bin/bash

echo "Groups: "

for gid in $(id -G); do
    group=$(cat /etc/group |
            grep ":$gid:" |
            cut -f 1 -d":")
    echo "  $group";
done
```

```
$ ./groups
Groups:
  rimsa
  adm
  cdrom
  sudo
  dip
  plugdev
  lpadmin
  sambashare
```

for

- Uma versão alternativa do comando **for** permite navegar na lista de argumentos da linha de comando

```
for loop-index; do
    commands
done
```

- O mesmo comportamento pode ser simulado pelo comando a seguir, onde "$@" expande em argumentos em linha de comando entre aspas ("$1", "$2", ...)

```
for loop-index in "$@"; do
    commands
done
```

Exemplo

- Listando os parâmetros em linha de comando

candies

```
#!/bin/bash

for arg; do
    echo "$arg"
done
```

```
$ ./candies candy gum chocolate
candy
gum
chocolate
```

for ((expr ; expr ; expr))

- Uma outra versão do for com sintaxe similar àquelas de linguagens de programação como C pode ser usada

```
for (( expr1 ; expr2 ; expr3 )); do
    commands
done
```

  - A expressão *expr1* é avaliada antes da primeira iteração;
  - Os comandos (*commands*) são executados repetidamente enquanto o valor de *expr2* avaliar em verdadeiro
  - Após cada iteração, a expressão *expr3* é avaliada

Múltiplas expressões podem ser avaliadas se separadas por vírgula

Dica: um loop infinito pode ser simulado usando a sintaxe: ((; ;))

Exemplos

a) Listar três números aleatórios

rand3
```
#!/bin/bash

for (( i=1; i <= 3; i++ )); do
    echo "Random $i: $RANDOM"
done
```

```
$ ./rand3
Random 1: 26851
Random 2: 26233
Random 3: 26699
```

b) Mostrar três número com passos de tamanho 5 a partir de 10

by3
```
#!/bin/bash

for ((i=1, j=10; i <= 3 ; i++, j=j+5)); do
    echo "Number $i: $j"
done
```

```
$ ./by3
Number 1: 10
Number 2: 15
Number 3: 20
```

c) Mostrar números de 1 até o infinito com espera linear

inf
```
#!/bin/bash

for (( i=1 ; ; )); do
   sleep $i
   echo "Number: $((i++))"
done
```

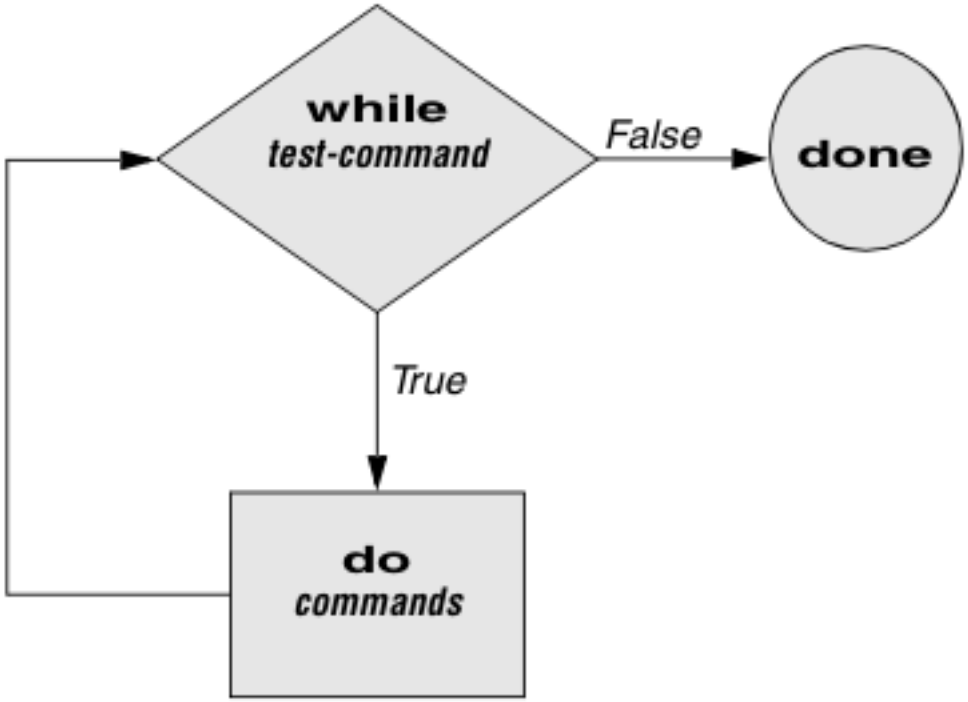
```
$ ./inf
Number: 1
Number: 2
^C
```

COMANDO `while`

while

- A estrutura de controle de repetição **while** possui a seguinte sintaxe

```
while test-command; do
    commands
done
```



Exemplo

- Imprimir dígitos de 0 a 9 sem pular linha usando o comando **`while`**

count

```
#!/bin/bash

number=0
while [ "$number" -lt 10 ]; do
    echo -n "$number"
    ((number +=1))
done
echo
```

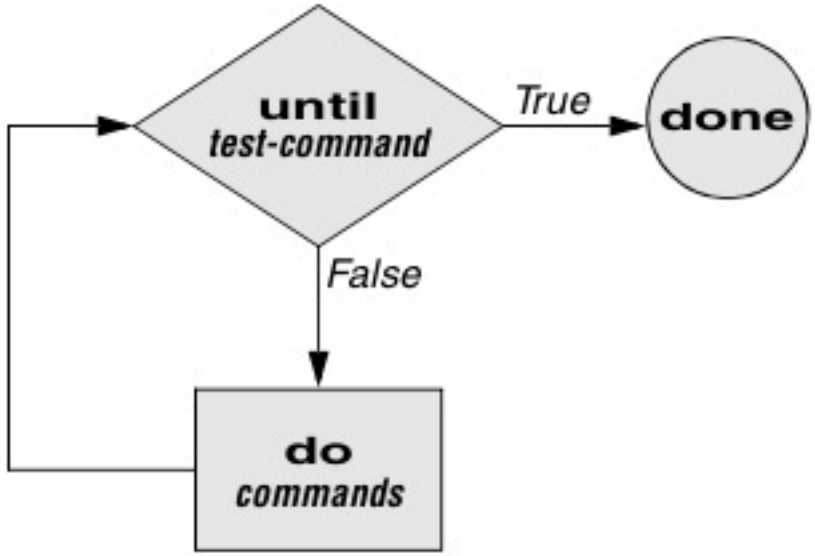
```
$ ./count
0123456789
```

COMANDO `until`

until

- O comando **until** tem estrutura muito similar ao comando **while** - a diferença está apenas na forma que é feita o teste condicional

```
until test-command; do
    commands
done
```



Exemplo

- Usar o comando **`until`** para adivinhar um nome secreto

guess

```
#!/bin/bash

secretname=zach
name=

echo "Try to guess the secret name!"
echo

until [ "$name" = "$secretname" ]; do
  echo -n "Your guess: "
  read name
done

echo "Very good"
```

```
$ ./guess
Try to guess the secret name!

Your guess: helen
Your guess: barbara
Your guess: rachel
Your guess: zach
Very good
```

Um Exemplo Real

- Travar a tela até que a senha correta, escolhida previamente pelo usuário, seja especificada

locktty

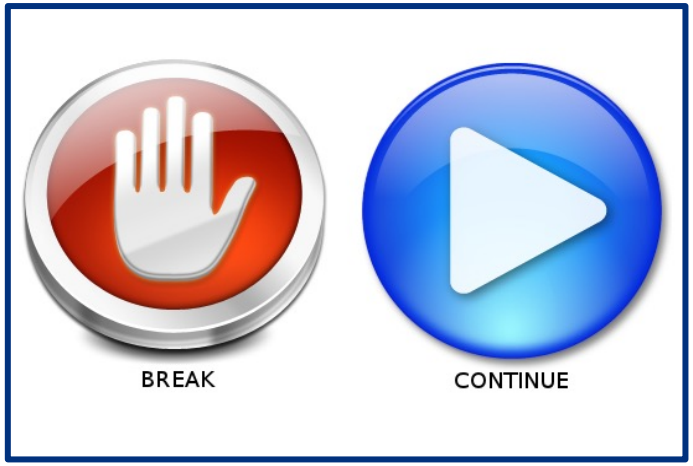
```
#!/bin/bash

trap '' 1 2 3 18
stty -echo

echo -n "Key: "
read key_1
echo
echo -n "Again: "
read key_2
echo

key_3=
if [ "$key_1" = "$key_2" ]; then
    tput clear

    until [ "$key_3" = "$key_2" ]; do
        read key_3
    done
else
    echo "locktty: keys do not match" 1>&2
fi

stty echo
```

Dicas:

1) use o comando **`trap`** para inibir sinais (control+c, ...)
2) consulte o manual de **`signal`** para obter o código de cada sinal

Cuidado: se esquecer a senha desse programa será preciso matar o processo (kill) usando um outro terminal

SEQUENCIADORES `break` E `continue`



`break` e `continue`

- Os comandos **`break`** e **`continue`** podem interromper o fluxo das estruturas de repetição **`for`**, **`while`** e **`until`**

`break`: transfere o controle para o comando imediatamente após o comando **`done`** do laço

`continue`: transfere o controle para o comando imediatamente após o comando **`do`** do laço

Exemplo

- Um laço de 1 a 10 que imprime apenas os números entre 4 e 8

flow

```
#!/bin/bash

for index in {1..10}; do
    if [ $index -le 3 ]; then
        echo "continue"
        continue
    fi

    echo $index

    if [ $index -ge 8 ] ; then
        echo "break"
        break
    fi
done
```

```
$ ./flow
continue
continue
continue
4
5
6
7
8
break
```

COMANDO `select`

select

- O comando **select** (baseado no encontrado em Korn Shell) exibe um menu, atribui o conteúdo do item escolhido à variável definida pelo usuário e executa uma série de comandos; a sintaxe é

```
select varname [in arg...]; do
    commands
done
```

Assim como no **for**, se in não for especificado, será utilizada a expansão **"$@"**

select

- Por exemplo, um comando **select** que começa com

```
select fruit in apple banana blueberry kiwi
                orange watermelon STOP
```

exibe um menu como

```
1) apple  3) blueberry  5) orange        7) STOP
2) banana 4) kiwi       6) watermelon
```

- O comando **select** usa as variáveis **LINES** (padrão: 24) e **COLUMNS** (padrão: 80) que especificam o tamanho do display; com COLUMNS definido com 40 o menu é exibido como

```
1) apple       5) orange
2) banana      6) watermelon
3) blueberry   7) STOP
4) kiwi
```

`select`

- Após exibir o menu o comando **`select`** exibe o valor da variável **PS3** (*select prompt*), onde o valor padrão é **#?**, mas que normalmente é modificado para um valor mais apropriado

- Ao entrar com um número válido (na faixa do menu), o comando **`select`** atribui a variável *varname* o argumento correspondente ao número do item escolhido; caso contrário, o comando **`select`** define *varname* como **`null`**
  - De qualquer forma, o comando **`select`** coloca o número do item escolhido na variável **REPLY**

Apertar *enter* sem definir uma opção causa a reexibição do menu com o *select prompt*

`select`

- Depois, executa-se os comandos especificados no corpo do comando **`select`**

- A estrutura de fluxo **`select`** continua exibindo um *select prompt* (PS3) e executa os comandos definidos no seu corpo até que o fluxo seja interrompido explicitamente - tipicamente por um comando como **`break`** (sair do laço) ou **`exit`** (sair do script)

Exemplo

- Um laço para escolha de frutas

favorite

```
#!/bin/bash

PS3="Choose your favorite fruit: "

select FRUIT in apple banana blueberry kiwi \
                orange watermelon STOP; do
    if [ "$FRUIT" == "" ]; then
        echo "Invalid entry"
        echo
        continue
    elif [ $FRUIT = STOP ]; then
        echo "Thanks for playing!"
        break
    fi

    echo "You chose $FRUIT as your favorite"
    echo "That is choice number $REPLY"
    echo
done
```

```
$ export COLUMNS=40
$ ./favorite
1) apple        5) orange
2) banana       6) watermelon
3) blueberry    7) STOP
4) kiwi
Choose your favorite fruit: 3
You chose blueberry as your favorite
That is choice number 3

Choose your favorite fruit: 99
Invalid entry

Choose your favorite fruit: 7
Thanks for playing!
$
```

ISSO É TUDO, PESSOAL!