[MAC0211] Laboratório de Programação I
Aula 12
Programação Bash
Arquivos

Kelly Rosa Braghetto

DCC–IME–USP

11 de abril de 2013

[Aula passada] Construtores de laços – comando `while`

- Executa os *comandos conseguintes* enquanto os *comandos de teste* possuírem um status de saída zero.
  O status devolvido pelo *while* é o status de saída do último comando executado dos *comandos conseguintes* (ou zero, caso nenhum tenha sido executado).

```
while {comandos de teste}; do
    {comandos conseguintes}
done
```

Lembrete: Status de saída zero = sucesso na execução do comando ou programa

[Aula passada] Comando `while`

Exemplo

```
#!/bin/bash

CONTADOR=0
while [ $CONTADOR -lt 10 ]; do
    echo O contador vale $CONTADOR
    let CONTADOR=CONTADOR+1
done
```

[Aula passada] Alguns comandos para expressões lógicas

Expressão	**Verdadeira se ...**
[-a ARQ]	o arquivo ARQ existe
[-d DIR]	o diretório DIR existe
[-z STRING]	o comprimento de STRING é zero
[-n STRING]	o comprimento de STRING não é zero
[STRING1 = STRING2]	as strings são iguais
[STRING1 != STRING2]	as strings são diferentes
[STRING1 < STRING2]	STRING1 é lexicograf. menor que STRING2
[NUM1 -eq NUM2]	o inteiro NUM1 é igual ao inteiro NUM2
[NUM1 -ne NUM2]	o inteiro NUM1 não é igual ao inteiro NUM2
[NUM1 -lt NUM2]	o inteiro NUM1 é menor que o inteiro NUM2
[NUM1 -le NUM2]	o inteiro NUM1 é menor ou igual a NUM2
[NUM1 -gt NUM2]	o inteiro NUM1 é maior que o inteiro NUM2
[NUM1 -ge NUM2]	o inteiro NUM1 é maior ou igual a NUM2
[! EXPR]	EXPR é uma expressão falsa
[EXPR1 -a EXPR2]	ambas as expressões são verdadeiras
[EXPR1 -o EXPR2]	ao menos uma das expressões é verdadeira

[Aula passada] Construtores de laços – comando `until`

- Executa os *comandos conseguintes* enquanto os *comandos de teste* possuírem um status de saída diferente de zero.
  O status devolvido pelo *until* é o status de saída do último comando executado dos *comandos conseguintes* (ou zero, caso nenhum tenha sido executado).

```
until {comandos de teste}; do
    {comandos conseguintes}
done
```

[Aula passada] Comando until

Exemplo

```
#!/bin/bash

CONTADOR=20
until [ $CONTADOR -lt 10 ]; do
    echo O contador vale $CONTADOR
    let CONTADOR=CONTADOR-1
done
```

[Aula passada] Construtores de laços – comando `for`

- Expande *palavras* e executa os *comandos conseguintes* uma vez para cada membro da lista resultante da expansão, sendo que a variável *nome* contém o membro atual.
  O status devolvido pelo *for* é o status de saída do último comando executado dos *comandos conseguintes* (ou zero, caso nenhum tenha sido executado).

```
for {nome} in {palavras ... }; do
  {comandos conseguintes}
done
```

Comando for

[Aula passada] Exemplo 1 – percorre os arquivos do diretório atual

```
#!/bin/bash

for i in $( ls ); do
    echo item: $i
done
```

[Aula passada] Exemplo 2 – lista os números de 1 a 10

```
#!/bin/bash

for i in `seq 1 10`; do

    echo $i
done
```

Construtores condicionais – comando *if*

- Os *comandos de teste* são executados e se o status de retorno for zero, os *comandos conseguintes* do *if* são executados. Caso o status for diferente de zero, os comandos de teste do *elif* são executados e, se o status de retorno for zero, os comandos conseguintes correspondentes são executados. Se o *else* está presente e os comandos das cláusulas do *if* e do *elif* tiverem um status de saída diferente de zero, então os comandos conseguintes alternativos são executados.

```
if {comandos de teste}; then
    {comandos conseguintes}
[elif {mais comandos de teste}; then
    {mais conseguintes}]
[else
    {conseguintes alternativos}]
fi
```

Comando if-elif-else

Exemplo – verifica se a variável de ambiente NUMERO contém um número

```
#!/bin/sh

if [ $NUMERO -gt 0 ]; then
    echo "$NUMERO e' positivo"
elif [ $NUMERO -lt 0 ]; then
    echo "$NUMERO e' negativo"
elif [ $NUMERO -eq 0 ]; then
    echo "$NUMERO e' zero"
else
    echo "Ooops! $NUMERO nao e' um numero; "
fi
```

Construtores condicionais – comando *case*

Exemplo – *script* que caracteriza animais

```
#!/bin/bash

echo -n "Digite um tipo de animal: "
read ANIMAL
echo -n "O $ANIMAL possui "
case $ANIMAL in
    cavalo | cachorro | gato) echo -n "quatro";;
    homem | canguru ) echo -n "duas";;
    *) echo -n "um número desconhecido de";;
esac
echo " pernas."
```

Construtores condicionais – comando *select*

- Cria um menu com as entradas passadas para o comando
- O índice da opção selecionada é armazenado na variável REPLY
- O *select* é repetido até que o comando *break* seja executado

Exemplo

```
select ARQ in *; do
    echo você selecionou o arquivo texto $ARQ \($REPLY\)
    break;
done
```

Construtores condicionais – comando *select*

Outro exemplo

```
#!/bin/bash

OPCOES="Hello Sair"
select opt in $OPCOES; do
    if [ "$opt" = "Sair" ]; then
        echo Tchau
        exit
    elif [ "$opt" = "Hello" ]; then
        echo Hello World
    else
        echo Opcao invalida
    fi
done
```

Argumentos passados via linha de comando

- São acessados via parâmetros posicionais
- **$#** – número de argumentos passados (sem contar o nome do *script*)
- **$0** – nome do script
- **$1**, **$2**, **$3**, ... – parâmetros 1, 2, 3, ...

Exemplo – Script que diz oi para o usuário

```
#!/bin/bash

if [ $# -ne 1 ]; then
    echo Uso: $0 [nome]
else
    echo Oi, $1
do
```

Funções e variáveis locais

- Estrutura para a criação de funções:

  `function minha_funcao { meu_codigo }`
- Usa-se a palavra-chave *local* para a criação de variáveis locais
- O status de saída de uma função é o status de saída do último comando executado nela

Exemplo

```
#!/bin/bash
HELLO=Hello

function fhello {
    local HELLO=World
    echo $HELLO
}
echo $HELLO             # deve mostrar "Hello"
fhello                  # chama fhello, que deve mostrar "World"
echo $HELLO             # deve mostrar "Hello" novamente
```

Passagem de parâmetros para funções

- Parâmetros passados para funções são tratados de forma semelhante aos parâmetros passados para o *script*

Exemplo

```
#!/bin/bash

function soma {
    echo $1+$2 = $[$1 + $2]
}

soma 23 75   # chama a funcao soma passando dois parametros
```

Misturando um pouco de cada coisa...

Exemplo – *script* renomeador de arquivos

```
#!/bin/bash
# renomeia

STRDE=$1
STRPARA=$2
for i in $( ls *$STRDE* ); do
    ORIGEM=$i
    DESTINO=$(echo $i | sed -e "s/$STRDE/$STRPARA/")
    mv $ORIGEM $DESTINO
done
```

Exemplo de uso do *script*: `./renomeia teste meuteste`
O comando substitui todas as ocorrências da palavra "teste" em nomes de arquivos do diretório por corrente pela palavra "meuteste".

Obs.: Há uma versão mais "sofisticada" de *script* para renomear arquivos nesta página: `http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-12.html#ss12.3`

Esclarecendo uma discussão levantada na aula passada...

O uso do "./" para executar um *script* do diretório corrente

- Para executar um script `meu_script` que se encontra no diretório corrente, geralmente usamos: `./meu_script [parâmetros]`
- A intenção é dizer que queremos executar no *shell* um comando que não é *builtin* e nem está no caminho de busca (*path*). Por isso, precisamos dizer para o *shell* explicitamente sua localização, que, no caso de um arquivo no diretório corrente, é denotada pelo "./".
- Podemos adicionar o diretório corrente (.) no *path*, mas isso não é uma boa ideia sob o ponto de vista de segurança. Se algum *script* malicioso tiver sido "implantado" no diretório corrente e esse *script* tiver o nome de um comando que usamos com frequência (por exemplo, o `ls`), então o *script* malicioso seria executado em substituição do comando verdadeiro caso o diretório corrente estivesse no *path*.

Esclarecendo uma discussão levantada na aula passada...

O operador *source* ou "." (ponto) [1]

- Sintaxe:
  `source nome_do_arquivo` [parâmetros]
  ou
  `. nome_do_arquivo` [parâmetros]
- Funcionalidade: carrega os comandos armazenados em `nome_do_arquivo` e os executa no contexto do *shell* corrente (ou seja, as variáveis criadas ou modificadas pelo *script* vão permanecer disponíveis depois que a execução do *script* terminar).

[1] Mais detalhes em: `http://ss64.com/bash/period.html`

Arquivos

Classificações

Texto × Binário; Dados × Executável

- **Arquivo Texto** – é estruturado como uma sequência de linhas de texto. De forma geral, podem conter apenas caracteres "imprimíveis", tabulações horizontais e quebras de linhas.
- **Arquivo Binário** – podem conter qualquer tipo de dados, codificados em formato binário, para propósitos de armazenamento ou processamento computacional. Frequentemente, contém bytes que não devem ser interpretados como caracteres de texto.

Arquivos

Arquivo Texto

- Caracteres de quebra de linha
  Carriage return (CR) [ASCII 13] e Line Feed (LF) [ASCII 10]
- Cada SO usa uma quebra de linha diferente
  - Windows: CR+LF
  - Unix: LF
  - Mac: CR

Arquivos

Classificações

Texto × Binário; Dados × Executável

- **Arquivo de Dados** – contém informações que são entrada ou saída de programas de computadores.
- **Arquivo Executável** – contém código ou instruções a serem executadas.

Arquivos de dispositivos

O Unix e seus derivados são sistemas orientados a arquivos:

- Os dispositivos periféricos também são tratados como um tipo especial de arquivo
- Esses arquivos especiais possibilitam que programas interajam com qualquer dispositivo por meio de chamadas ao sistema padronizadas para operações de E/S
- Exemplos de arquivos de dispositivos de entrada e saída:
  - impressora – /dev/lp0
  - console – /dev/console
  - hard disk – /dev/sd[x] ou /dev/hd[x]...
  - cdrom – /dev/cdrom
  - /dev/null [não está associado a um dispositivo físico!]
- Arquivos de dispositivos podem ser de três tipos: *de caracter*, *de bloco* e *pseudo-dispositivo*

"Brincando" com os arquivos de dispositivos

Faça este teste e observe o resultado:

- em um terminal, executar o comando `tty`. O resultado será algo como `/dev/pts/3`.
- em outro terminal, executar o comando `cat arq >/dev/pts/3` (onde `arq` é um arquivo texto presente no diretório atual).
- Executar também *cat >/dev/pts/3*. Digite caracteres e pressione [Ctrl-D] para efetuar a "transmissão". [CRTL+D] em uma linha vazia encerra a "transmissão".

"Curiosidade" sobre o cat

- Quando nenhum arquivo de entrada é passado para o cat, ele lê caracteres da entrada padrão até que as teclas **CRTL-D** sejam pressionadas em uma linha vazia.
- Exemplo: o comando abaixo cria um arquivo de nome "meu_arq.txt", gravando nele tudo o que o usuário digitar até que **CTRL-D** seja pressionado em uma linha vazia

```
cat > meu_arq.txt
```

Lição de casa

Leitura de texto sobre o projeto de interfaces (de bibliotecas)

- Capítulo 4 – *Interfaces*, do livro *The Practice of Programming*, de B.W. Kernighan e R. Pike

  (A cópia digital do capítulo está no Paca, junto com o material da aula de hoje.)

Bibliografia e materiais recomendados

- Manual do bash
  `http://www.gnu.org/software/bash/manual/bashref.html`
- *HowTo* de programação no Bash
  `http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html`
- Arquivos e *Streams* em C – manual de referência da libc
  `http://www.gnu.org/software/libc/manual/html_mono/libc.html#I_002f0-Overview`
- Filosofia do Unix e *pipes* – livro: *The Art of Unix Programming*, de E.S. Raymond
  `http://www.catb.org/esr/writings/taoup/html/`
- Notas das aulas de MAC0211 de 2010, feitas pelo Prof. Kon
  `http://www.ime.usp.br/~kon/MAC211`

Cenas dos próximos capítulos...

Nas próximas aulas, veremos um novo assunto

- Modularização de programas escritos em C
- Gerenciamento de compilação de programas e bibliotecas com ferramentas