Universidade de São Paulo
Brasil

Aula 10
programação em bash

MAC0216 - Técnicas de Programação I

Professores: Alfredo, Daniel, Fabio e Kelly

Departamento de Ciência da Computação
Instituto de Matemática e Estatística

1.
Programação em bash

Comandos compostos

- São os construtores de programação shell
- Começam com uma palavra reservada ou operador de controle e terminam com uma palavra reservada ou operador de controle correspondente
- Qualquer redirecionamento associado a um comando composto aplica-se a todos os seus comandos internos (a menos que sejam explicitamente sobrescritos)
- Categorias: comandos de laços, comandos condicionais, mecanismos de agrupamento

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

Comando while

Exemplo

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

Alguns comandos para expressões lógicas

Expressão	Verdadeira se ...
[-e 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

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
```

Comando until

Exemplo

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

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

Exemplo 1 – percorre os arquivos do diretório atual

```
#!/bin/bash
for i in $( ls ); do
    echo item: $i
done
```

Exemplo 2 – lista os números de 1 a 10

```
#!/bin/bash
for i in `seq 1 10`; do
    echo $i
done
```

Comando for

Exemplo 3 – percorre os arquivos do diretório atual (funcionando corretamente quando houver espaços)

```
#!/bin/bash
# IFS é a variável que guarda os caracteres
# separadores de itens
IFS='
'
for i in $( ls ); do
    echo item: $i
done
```

Comando for

Exemplo 4 – percorre as linhas de um arquivo

(funcionando corretamente quando houver espaços)

```
#!/bin/bash

export IFS='
'
for linha in `cat arquivo`; do
    echo item: $linha
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/bash
if [ $NUMERO -gt 0 ]; then
    echo "$NUMERO eh positivo"
elif [ $NUMERO -lt 0 ]; then
    echo "$NUMERO eh negativo"
elif [ $NUMERO -eq 0 ]; then
    echo "$NUMERO eh zero"
else
    echo "Ooops! $NUMERO nao eh 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 numero 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
fi
```

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 corrente pela palavra "`meuteste`". Obs.: O comando `rename` pode ser usado para essa tarefa.

Material recomendado

- Guia avançado de bash script

  https://tldp.org/LDP/abs/html/abs-guide.html