

[MAC0211] Laboratório de Programação I
Aula 11
Interpretadores de Comandos (*Shells*)
Programação Bash

Kelly Rosa Braghetto

DCC-IME-USP

9 de abril de 2013

Outros tipos de execução de comandos no bash

- ▶ **Execução em *background*** – faz com que um comando seja executado assincronamente, em um *subshell*. Em uma execução em background, o *shell* não espera que a execução do comando disparado termine. É feita quando o caracter de controle ‘&’ é usado no final do comando. Exemplo:

```
sort mac211.txt > saida.txt &
```

- ▶ **Execução sequencial** – comandos separados por ‘;’ são executados sequencialmente. Exemplo:

```
sort mac211.txt > saida.txt ; cat saida.txt
```

Outros tipos de execução de comandos no bash

- ▶ **Execução paralela** – feita por meio do comando *GNU Parallel* permite que comandos sejam executados paralelamente. Exemplos:

```
cat mac211.txt | parallel -k echo Aluno:
```

Mostra o conteúdo `mac211.txt` de incluindo o prefixo “Aluno:” em todas as linhas. A opção `-k` garante que a ordem das linhas seja mantida.

```
cat mac211.txt | parallel -k echo {} ' -Aluno'
```

Mostra o conteúdo `mac211.txt` de incluindo o sufixo “-Aluno” em todas as linhas.

Execução de listas de comandos no bash

- ▶ **Lista E** – é uma sequência de um ou mais comandos separados pelo operador de controle '&&', como em:

```
comando1 && comando2
```

O comando2 é executado se e somente se a execução de comando1 terminou com sucesso (ou seja, status de saída = 0).

- ▶ **Lista OU** – é uma sequência de um ou mais comandos separados pelo operador de controle '||', como em:

```
comando1 || comando2
```

O comando2 é executado se e somente se a execução de comando1 não terminou com sucesso (ou seja, status de saída $< > 0$).

Comandos *builtin* do bash

Comandos *builtin* são comandos contidos no próprio *shell*, ou seja, são comandos que o *shell* executa diretamente, sem invocar outros programas. Alguns comandos *builtins* do bash:

- ▶ `history`
Exibe uma lista dos comandos já executados
- ▶ `pwd`
Mostra o caminho do diretório atual
- ▶ `alias <nome>=<comando>`
Cria um “apelido” para um comando

Comandos do `bash` acionados via [combinação de] teclas

- ▶ flechas verticais – percorre histórico de comandos
- ▶ flechas horizontais – posicionamento na linha corrente
- ▶ TAB – completa comando que começou a ser digitado (até onde dá sem ambiguidade)
- ▶ TAB TAB – mostra lista das opções de comandos possíveis para completar o que já começou a ser digitado (se houver várias)
- ▶ [CTRL-C] – cancela a execução do programa corrente
- ▶ [CTRL-D] – gera código de final de arquivo (fechando o terminal); também fim de sessão de login

Comandos do `bash` acionados via [combinação de] teclas

- ▶ [CTRL-Z] – suspende (“congela”) o programa correntemente em execução.

O comando `jobs` lista os programas que estão suspensos e seus respectivos números.

Para retomar a execução do programa suspenso, há duas possibilidades de comandos:

- ▶ `fg %<num. do prog. suspenso>`
ou simplesmente
`%<num. do prog. suspenso>`
Retoma a execução do programa suspenso em primeiro plano (modo síncrono)
- ▶ `bg %<num. do prog. suspenso>`
ou simplesmente
`%<num. do prog. suspenso> &`
Retoma a execução do programa suspenso em *background* (modo assíncrono)

A pilha de diretórios do bash

O `bash` permite manter em uma pilha os diretórios visitados mais recentemente. Os seguintes comandos *builtin* manipulam essa pilha:

- ▶ `pushd <dir>`
Empilha o diretório atual e depois faz um `cd` para `dir`
- ▶ `popd <opções>`
Desempilha um diretório e depois faz um `cd` para ele
- ▶ `dirs <opções>`
Mostra informações sobre o status atual da pilha de diretórios

Arquivos no UNIX – permissões de acesso

Todo objeto do sistema de arquivos do UNIX (arquivo ou diretório):

- ▶ pertence a um usuário e a um grupo
- ▶ possui 3 conjuntos de permissões, que definem os modos de acesso permitidos:
 - ▶ ao usuário dono do objeto (*owner*, *u*)
 - ▶ aos membros do grupo do objeto (*group*, *g*)
 - ▶ a todos os usuários do sistema (*others*, *o*)

Há três modos de acesso possíveis

- ▶ *r* – leitura (de *read*)
- ▶ *w* – escrita (de *write*)
- ▶ *x* – execução (de *execution*)

Arquivos no UNIX – permissões de acesso

O comando `chmod` altera as permissões de acesso de objetos do sistema de arquivos. O comando pode ser usado em dois modos: *octal* ou *simbólico*.

▶ `chmod <modo> <arquivo(s)>`

onde modo é um número de três dígitos obtido por meio da tabela abaixo. Exemplo:

```
chmod 664 arq_compartilhado
```

(o comando atribui as permissões de leitura e escrita para o dono e o grupo, e somente leitura para outros usuários).

	dono	grupo	outros
	rxw	rxw	rxw
0 -	000	000	000
1 -	001	001	001
2 -	010	010	010
3 -	011	011	011
4 -	100	100	100
5 -	101	101	101
6 -	110	110	110
7 -	111	111	111

Arquivos no UNIX – permissões de acesso

O comando `chmod` altera as permissões de acesso de objetos do sistema de arquivos. O comando pode ser usado em dois modos: *octal* ou *simbólico*

- ▶ `chmod <referência> <operador> <modo> <arquivo(s)>`

Referência: u, g ou o (e suas combinações)

Operador: =, + (acrescenta) ou - (exclui)

Modo: r, w ou x (e suas combinações)

Exemplo:

```
chmod ug=rw,o=r arq_compartilhado
```

Scripts

- ▶ *Scripts* são arquivos contendo comandos shell
- ▶ Para que esses arquivos tenham status de novos comandos, eles devem ser executáveis. Para transformar um *script* em executável:
`chmod u+x meu_script`
- ▶ *Scripts* podem ser iniciados por `#!` – o *shebang* (também chamado de *hashbang*, entre outros) : diretiva que indica o caminho do *shell* que deve ser carregado para executar (interpretar) o *script*
- ▶ O caracter `#` sozinho delimita o início de uma linha de comentário

Exemplo – programa “Hello World”

```
#!/bin/bash  
echo Hello World!      # imprime "Hello World" na saída padrão
```

Variáveis de ambiente

- ▶ Variáveis em bash não possuem tipo
- ▶ Elas podem conter números, caracteres ou cadeias de caracteres
- ▶ Elas não precisam ser declaradas; para criar uma variável, basta atribuir um valor a ela
- ▶ Para recuperar o valor armazenado em uma variável, colocar um '\$' em frente ao seu nome

Exemplo

```
STR='Hello World!'  
echo $STR
```

Obs.: Note que na definição da variável, não pode haver espaços nem antes e nem depois do sinal de “=”

Variáveis + expressões

Exemplo – *script* que faz *backup* do *home*

```
#!/bin/bash
ARQ_SAIDA=/var/meu-backup-$(date +%d%m%Y).tgz
tar -czf $ARQ_SAIDA /home/usuario/
```

- ▶ A expressão `$(comando)` executa *comando* e captura o resultado dele
- ▶ `$(date +%d%m%Y)` – expressão que executa o comando `date`, gerando uma *string* com a data atual no formato dia-mês-ano
- ▶ o nome do arquivo com o *backup* (compactado) dos dados do diretório *home* do usuário possui como sufixo a data em que o arquivo foi criado

Variáveis de ambiente definidas com export

- ▶ Uma variável de ambiente também pode ser definida com o comando `export`. Exemplo:
`export PATH=/usr/bin`
- ▶ Depois de definida com o `export`, uma variável fica “visível” nos scripts ou programas executados posteriormente

Exemplo

Script “digaoi”

```
#!/bin/bash  
echo Oi, $USUARIO
```

Execução

```
$ export USUARIO=Kelly  
$ ./digaoi  
Oi, Kelly
```

Variáveis de ambiente “famosas”

- ▶ **PATH** – caminhos para a busca de programas
- ▶ **PWD** – diretório corrente
- ▶ **SHELL** – shell padrão
- ▶ **CDPATH** – diretórios usados como base para o comando *cd*
- ▶ **JAVA_HOME** – diretório de instalação do Java

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

```
while {comandos de teste}; do
    {comandos consequentes}
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 ...
<code>[-a ARQ]</code>	o arquivo ARQ existe
<code>[-d DIR]</code>	o diretório DIR existe
<code>[-z STRING]</code>	o comprimento de STRING é zero
<code>[-n STRING]</code>	o comprimento de STRING não é zero
<code>[STRING1 = STRING2]</code>	as strings são iguais
<code>[STRING1 != STRING2]</code>	as strings são diferentes
<code>[STRING1 < STRING2]</code>	STRING1 é lexicograf. menor que STRING2
<code>[NUM1 -eq NUM2]</code>	o inteiro NUM1 é igual ao inteiro NUM2
<code>[NUM1 -ne NUM2]</code>	o inteiro NUM1 não é igual ao inteiro NUM2
<code>[NUM1 -lt NUM2]</code>	o inteiro NUM1 é menor que o inteiro NUM2
<code>[NUM1 -le NUM2]</code>	o inteiro NUM1 é menor ou igual a NUM2
<code>[NUM1 -gt NUM2]</code>	o inteiro NUM1 é maior que o inteiro NUM2
<code>[NUM1 -ge NUM2]</code>	o inteiro NUM1 é maior ou igual a NUM2
<code>[! EXPR]</code>	EXPR é uma expressão falsa
<code>[EXPR1 -a EXPR2]</code>	ambas as expressões são verdadeiras
<code>[EXPR1 -o EXPR2]</code>	ao menos uma das expressões é verdadeira

Construtores de laços – comando `until`

- ▶ Executa os *comandos consequentes* 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 consequentes* (ou zero, caso nenhum tenha sido executado).

```
until {comandos de teste}; do
    {comandos consequentes}
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 consequentes* 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 consequentes* (ou zero, caso nenhum tenha sido executado).

```
for {nome} in {palavras ... }; do
    {comandos consequentes}
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
```

Bibliografia e materiais recomendados

- ▶ Manual do bash
<http://www.gnu.org/software/bash/manual/bashref.html>
- ▶ Guia para iniciantes do bash
<http://www.tldp.org/LDP/Bash-Beginners-Guide/html/>
- ▶ *HowTo* de programação no Bash
<http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html>
- ▶ *Bash by example* (Partes 1 e 2)
<http://www.ibm.com/developerworks/linux/library/l-bash/index.html>
<http://www.ibm.com/developerworks/linux/library/l-bash2/index.html>
- ▶ Notas das aulas de MAC0211 de 2010, feitas pelo Prof. Kon
<http://www.ime.usp.br/~kon/MAC211>

Cenas dos próximos capítulos...

Mais sobre programação Bash

- ▶ Comandos condicionais
- ▶ Passagem de parâmetros via linha de comando
- ▶ Funções

Mudando de assunto...

- ▶ Arquivos
- ▶ Filtros