Shell Scripting

Laboratório de Sistemas Operacionais

Prof. MSc. João Tavares

O que é Shell Scripting?

- Princípio básico:

Encadear/orquestrar a execução de vários comandos Unix para realizar uma tarefa

- Utilização:

Desenvolver rotinas para ajudar a automatizar atividades do dia-a-dia a partir de programas mais simples pré-existentes

A linguagem Shell Script

- Comandos Unix são construções válidas da linguagem

- Adicionalmente, possui estruturas importantes (típicas) de linguagens de programação de alto nível
  – Variáveis,
  – Funções,
  – Execução condicional (if, case),
  – Laços de repetição (for, while),
  – etc.

Programa Shell Script

- Qualquer sequência de comandos do Unix armazenados em um arquivo
  - Tudo que se pode digitar no prompt de comando é uma construção válida em um arquivo de script e vice-versa

- Arquivo pode ser executado
  - Arquivo serve de “memória” para não ser necessário re-digitar todos os comandos nas novas execuções do script

- Parecido como os arquivos batch (.bat) do DOS, porém mais poderosos

Scripts no Unix

1) Criar o arquivo texto para armazenar os comandos que compoe script
   - Usualmente, extensão reflete a linguagem em uso
   - Para scripts shell é adotada a extensão **.sh**

2) Alterar a permissão para executável

```
$ chmod u+x meuscript.sh
```

3) Chamar o arquivo no prompt do shell

```
$ /bin/bash ./meuscript.sh
```

Scripts no Unix

- UNIX prevê tratamento especial para programas do tipo script
  - Usualmente, a primeira linha do script inicia por **#!** e identifica o **interpretador** (shell) a ser usado

    Ex.: #!/bin/bash
  - Quando o script é executado, SO na verdade cria um **processo** daquele interpretador, fornecendo o próprio arquivo do script como parâmetro

- Preserva uma visão uniforme dos programas aos usuários
  - Script chamado na linha de comando pelo seu nome, como qualquer outro programa do sistema

    Ex.: /bin/bash/ ./meuscript.sh

Primeiro Shell Script

- Com o editor de texto de sua preferência (gedit, geany, nano, pico, cat >, vi, emacs, kwrite, etc) crie um arquivo *simples.sh* com o conteúdo abaixo:

```
#!/bin/bash
# Mostra algumas informações úteis no início do dia
date
echo Bem-vindo $USER
cal
last | head -3
```

- Ajuste as permissões do arquivo

```
$ chmod u+x simples.sh
```

- Execute

```
$ /bin/bash ./simples.sh
```

- O que o script *simples.sh* faz?

Primeiro Shell Script

Executando...

- Comandos não são mostrados, somente resultados
- Variavel $USER mostra o nome de login
  - Outras variaveis de ambiente: $HOME, $PATH
- Comentários são iniciados pelo caractere “#”
- Quando ocorre um erro, o script continua executando o próximo comando

Princípios fundamentais do Shell Script

- Manipulação de variáveis (de Ambiente)
  – Armazenam valores intermediários da computação
  – Podem afetar a forma como alguns comandos executam

- Composição de comandos (programas) do SO
  – Estruturas imperativas (if, for, etc.)
  – Filtros de dados (|) → saida gerada por um comando e tratada como entrada de um comando subsequente

- Expansão de expressões
  – Valor da expressão é **substituído antes** do comando ser avaliado

Variáveis

- São nomes que armazenam uma string
  – Iniciam com sublinhado (“_”) ou caractere alfabético, seguido de sequência alfanumérica

- São criadas na primeira atribuição

  variavel=valor *(sem espaço antes ou depois do “=“)*

- Expansão de variável → substitui a ocorrência do nome da variável pelo seu conteúdo
  – Nome da variável desejada prefixado por “$”
  – A expansão de uma variável pode substituir comandos
- Exemplo:

```
$ L='ls -l'
$ $L
```

Variáveis - Exemplo

- Script *variav.sh*:

```
#!/bin/bash
# Um exemplo com variáveis
nomearq=”/etc/passwd”
echo "Verifica a permissão no arquivo de senhas"
ls -l $nomearq
echo "Descobre quantas contas existem no sistema"
wc -l $nomearq
```

- Basta alterar o valor de $nomearq para a mudança ser propagada para todo o script

Variáveis de ambiente

- São variáveis que alteram o comportamento do ambiente Unix
  – Exemplos: $PS1 (prompt do shell), $HOME, $PATH, $USER, $PWD, etc.
  – Variáveis locais de um script podem ser exportadas, tornando-se variáveis de ambiente (públicas)

- Exemplo:

```
$ echo $HOME
$ echo $PATH
$ PATH=$PATH:$HOME
$ echo $PATH
$ export PATH
```

- Para listar as variáveis de ambiente → env

Processamento de Strings

- Aspas são utilizadas para proteger uma string que contenha espaços
  – Aspas duplas (“”) → habilita pré-processamento
    * Todas as expansões existentes dentro da string devem ser realizadas antes da utilização daquele conteúdo
  – Aspas simples (') → ignora pré-processamento do conteúdo da string

- Sequências de escape iniciadas por barra invertida (\) são utilizadas para forçar o uso de um caracter sem interpretá-lo

Expansões

- Expansão de variável → $V
  – Ocorrência é substituída pelo valor atualmente contido na variavel V

- Expansão aritmética → $((*exp*))
  – Ocorrência é substituida pela avaliação da string *exp*, considerando que a mesma é composta de variáveis e operadores aritméticos (+,-,*,/,%), lógicos e comparações
  – Não é necessário proteger *exp* com aspas duplas, mesmo que contenha espaços

Expansões

- Expansão de saída de comando → $(V)
  – Substituída pela saída gerada pela execução do comando *V*
    * *Observação:* **evitar usar** notação clássica com aspas invertidas devido a semelhança com aspas simples
  – `V` é visualmente muito semelhante a 'V'

- Expansão de status de comando → $?
  – Substituída pelo status de terminação do último comando executado a partir do script

Expansões

- Expansão de nome de arquivo →
  – Substituída pela lista de arquivos que satisfazem a expressão *glob* ('*man 7 glob*' para mais informações)

  – Alguns exemplos:

    * → todos os arquivos no diretorio atual, exceto os iniciados por '.'

    ab* → arquivos iniciados por *ab*

    /etc/f* → arquivos iniciados por f dentro de /etc/

    a[456]b → aquivos inciados por a, terminados por b e contendo entre estes 4, 5 ou 6

    a???b → arquivos com nomes de comprimento 5, iniciados por a e terminados com b

Expansões - Exemplos

- Teste os seguintes comandos no shell:

```
$ texto=”Testando...”
$ echo $texto
$ echo “$texto”
$ echo '$texto'
$ echo `$texto`
$ comando=$(date)
$ echo $comando
$ echo Hoje e $comando
$ echo Hoje e \”o dia\” de ganhar \$\$\$
$ echo 'Hoje e "o dia" de ganhar $$$'
$ echo este e um texto que\
> ocupa duas linhas
$ echo Seu diretorio atual e $(pwd)
```

- Quais os resultados?

Passagens de parâmetros

- Via variáveis de ambiente
  – Acessadas diretamente pelo nome de dentro do script

- Via parâmetros posicionais
  – Valores especificados logo após o nome do programa
    * Também chamados de parâmetros de linha de comando
  – São mapeados pelo shell para variáveis especiais

Acessando os parâmetros posicionais

- $0
  – O nome do script chamado (argumento zero)

- $1 … $9
  – Correspondem aos demais argumentos passados, em ordem
  – Nao existe $10, $11... → necessario usar shift

- $#
  – Contém o número de argumentos passados

- $*
  – Expansão para acessar todos os argumentos de uma única vez

Exemplo de acesso a parâmetros posicionais

- Exemplo: parametros.sh

```
#!/bin/bash
# Ilustra o uso de argumentos
echo "Primeiro argumento: $1"
echo "Terceiro argumento: $3"
echo "Número de argumentos: $#"
echo "Lista completa de argumentos: $*“
```

- shift → desloca argumentos para direita e elimina o primeiro argumento
  - Possibilita acesso aos argumentos após o $9
    - $1→ descartado; $2→$1; $3→$2; … $10→$9
- Altere o programa acima adicionando shift

Expressões aritméticas

- Obtidas através de expansões aritméticas → $((*e*))

  $ A=$((B+5*(D/3)))

- Na expressão:
  – variáveis não precisam ser precedidas de $
  – variáveis não definidas são inicializadas automaticamente com zero
  – aritmética é somente de inteiros
  – pode-se utilizar comparações

Expressões aritméticas

- Exemplos

```
$ valor=42
$ echo $((valor=valor-8))
$ echo $((6 * 45 / 9))
$ echo $(( valor > 100))
$ echo $(( valor < 100))
$ echo $(( valor == 42))
$ echo $(( valor != 42))
$ echo $(( valor != 100))
```

- Qual a saída gerada?

Status de terminação

- Código numérico

- Obrigatoriamente retornado pelo processo ao término de sua execução
  – Distinto da saída (texto) gerada pelo processo que é opcional

- Indica se o processo completou com sucesso ou fracasso sua a execução (tarefa)
  – zero → comando completado com sucesso
  – outro valor → comando terminou com erro (qual?)
    * Interpretação precisa do valor especificada na man page do comando

Status de terminação

- exit *status*
  – Termina a execução do script, opcionalmente informando um status de terminação
  – Se status omitido, propaga o valor em $?

- $? → status de terminação do comando mais recentemente executado
  – Em um pipeline, $? corresponde ao status de terminação do último comando do pipeline

Composição de comandos

- Filtragem de dados
  – operador pipe (|)

- Comportamento seletivo
  – if...then...else...fi
  – case...in...esac

- Repetição
  – for...do...done
  – while...do...done
  – until...do...done

Comando if

- Sintaxe (Não esquecer ';' e 'fi'):

```
if comando1; then
    bloco de comandos
elif comando2; then
    segundo bloco de comandos
else
    terceiro bloco de comandos
fi
```

- Comportamento
  - Executa o comando de teste especificado na condição do if e caso este retorne um código de terminação igual a zero (i.e., sucesso), executa o bloco do *if*, senão passa para o *else*
  - elif → else seguido de um novo if

Comando if

- Condição do **if** → pode ser qualquer comando do sistema que respeite a convenção do status de terminação

- Comando test
  – Combina em um único programa os testes mais frequentemente usados (não todos) em scripts
  – Visto que seu uso é frequente, o interpretador shell suporta uma sintaxe especial com colchetes ([])
  – Exemplo

```
$ test -r nomedearquivo
$ [ -r nomedearquivo ]
```

(Não esquecer espaço!)

Comando test

- Comparações sobre strings

Teste	Retorna sucesso (0) se...
string1 == string2	Strings são iguais
string1 != string2	Strings são diferentes
-n string	String não é nula (vazia)
-z string	String é nula

Comando test

- Comparações aritméticas

Teste	Retorna sucesso(0) se...
expr1 -eq expr2	Expressões são iguais
expr1 -ne expr2	Expressões são diferentes
expr1 -gt expr2	expr1 é maior que expr2
expr1 -ge expr2	expr1 é maior ou igual que expr2
expr1 -lt expr2	expre é menor que expr2
expr1 -le expr2	expr1 é menor ou igual a expr2
!expr	expr1 é falsa

Comando test

- Testes sobre arquivos e diretórios

Teste	Retorna sucesso(0) se...
-d arq	arq é um diretório
-e arq	arq existe
-f arq	arq é um arquivo regular
-r arq	O usuário pode ler o arquivo
-s arq	arq existe e tem tamanho maior que zero
-w arq	O usuário pode escrever no arquivo
-x arq	O usuário pode executar o arquivo

Exemplo de seleção

- Script *executa.sh*

```
#!/bin/bash
if [ $# -ne 1 ]; then
   echo "Use: $0 nomedoarquivo"
elif [ ! -e $1 ]; then
   echo "Arquivo não existe"
elif [ ! -x $1 ]; then
   echo "O arquivo $1 não é executável!"
   echo -n 'Alterando permissão... '
   chmod u+x $1
   echo 'feito.'
else
   echo "O arquivo $1 já é executável."
fi
```

- O que faz o script executa.sh?

Comandos while e until

- Sintaxe (Não esquecer ';' e ‘done'):

while *comando1*; **do**
bloco de comandos
done

until *comando2*; **do**
bloco de comandos
done

- Comportamento

– while → repete a execução do bloco de comandos enquanto o comando de teste for executado e retornar sucesso

– until → repete a execução do bloco de comandos até que o comando de teste seja executado e retorne sucesso

Exemplos de repetição

- Script *conta.sh*

```
#!/bin/bash
CONTADOR=0
while [ "$CONTADOR" -lt 10 ]; do
      echo "Contador em: $CONTADOR"
      CONTADOR=$((CONTADOR+1));
done
```

- Observações:
  - É saudavel sempre proteger as variáveis em testes com “”, para que o shell não interprete uma variável vazia como erro de sintaxe
  - Não esquecer o espaço antes do]
  - Não esquecer ';' depois do] se 'do' na mesma linha

Comando for

- Sintaxe (Não esquecer ';' e ‘done'):

  for v in *lista de valores*; **do**
  bloco de comandos
  done

- Comportamento
  – Para cada um dos valores especificado na lista, atribui o próximo valor à variável do laço (v) e executa o bloco de comandos
  – Lista de valores para a iteração
  * Os valores na lista são separados por espaço
  * Lista pode ser gerada pela expansão de variáveis, comandos, máscaras de arquivos

Exemplos de iteração com for

- Script *conta3.sh*

```
#!/bin/bash
for CONTADOR in $(seq 10); do
   echo "Contador em: $CONTADOR"
done
```

- Script *itens.sh*

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

- Script *itens2.sh*

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

Comando case

- Sintaxe (Não esquecer “esac”):

```
case valor in
expr1 | expr2 | expr3 )          (Caracter “|” funciona)
        bloco de comandos
        ;;                       (Cláusula termina com ;;)
    expr4 )
        bloco de comandos
        ;;
    *)
        bloco de comandos
        ;;
esac
```

- Comportamento

  - Compara, sequencialmente, o resultado da expansão de valor com cada uma das expressões definidas

    * Em caso de *match (casamento de strings),* executa o bloco de comandos associado e sai do case.

Comando case

- As comparações são baseadas em strings
- *valor* é, tipicamente, o resultado de uma expansão de variável
- Podem ser definidas tantas cláusulas quantas forem desejáveis
  – Dentro de uma cláusula, expressões alternativas podem ser declaradas separando-as com |
  – *exp1*, … *expn* → são strings, podendo ser constantes absolutas ou incluir caracteres coringa (*glob*)
    * → casa com qualquer coisa, logo é utilizado como um *else* (última clausula)

Exemplos de Seleção com case

- Script *comandos.sh*

```
#!/bin/bash
echo "Entre seu comando (who, list, ou cal)"
read comando
case "$comando" in
    who | cal )
        echo "Executando $comando..."
        $comando
        ;;
    list)
        echo "Executando ls..."
        ls
        ;;
    *)
        echo "Comando errado. As opções são: who, list, ou
        cal"
        ;;
esac
```

Acessando a Entrada Padrão

- read *varname*
  - Lê uma string da entrada padrão para uma variável
  - Ex.:

```
$ read v
$ echo $v
```

- Mais informacoes:
  - **help read**
  - **man bash**

Funções

- Duas possíveis sintaxes:
  function *nomefunc* { *bloco de comandos*… }
  nomefunc() { *bloco de comandos*… }

- Agrupa um bloco de comandos sob um nome
  – usada como outro programa qualquer dentro do script
  – pode receber parâmetros ($1 … $9) e acessar variáveis do ambiente
  – local → define variável local à função (default: global)
  – return → termina a execução da função
    * opcionalmente, define um status de terminação

Exemplo de função

- Exemplo

```
$ function sayhello {
    local name=$1
    echo “Hello $1”
    return 0
}
$ sayhello Fulano
```

- Mais informações → **man** bash ou **info** bash

Leituras complementares

- Na biblioteca...
  – NEVES, Julio Cezar. **Programação Shell Linux**. 7. ed. Rio de Janeiro: Brasport, 2008. 450 p.

- Em formato digital...
  – Advanced Bash Scripting Guide
  http://tldp.org/LDP/abs/html/
  – Linux Shell Scripting Tutorial
  http://www.freeos.com/guides/lsst/
  – Paginas de man e info do bash
  – Shell (artigo abrangente da Wikipedia)
  http://en.wikipedia.org/wiki/Shell_(computing)
  – Canivete Suíço do Shell (Bash)
  http://aurelio.net/shell/canivete/#ifwhilefor

Referências Bibliográficas

- Material originalmente elaborado por Prof. Cristiano Costa. Material autorizado e cedido pelo autor. Revisado e atualizado por Prof. Luciano Cavalheiro e posteriormente pelo Prof. João Tavares.