

Programação em Shell

Taisy Weber

Programação em shell

✓ Comandos básicos bash

LDP, manuais para usuários
manuais UNIX e Linux

✓ Programação em shell

Matthew & Stones cap 2

- pipe,
- redirecionamento de entrada e de saída,
- comandos,
- variáveis, estruturas de controle e condicionais
- criação de scripts

TSW

2

Apoio bibliográfico

Siever, E. *Linux in a Nutshell*. O'Reilly

- comandos: preferencialmente **man**, **info** e **LDP**
- programação em shell
 - Matthew, N; Stones, R. "Beginning Linux Programming", Wrox press, 1999 (2th Edition) - capítulo 2
 - » conferir programas e scripts em:
<http://www.wrox.com>
cópia no servidor gaea.inf.ufrogs.br pasta ProgLinux
 - Ball, B. **Usando Linux**. Campus
Capítulo 22. Rápida introdução. Adequado para quem não domina o inglês.

TSW

3

Shell

✓ processador de comandos

reuso de código

✓ linguagem de programação (interpretador)

- adequada para prototipação
- programação rápida e simples
- ideal para pequenos utilitários
- fácil manutenção, configuração
- grande portabilidade

mas os programas não
são muito eficientes

✓ controle de tarefas

TSW

4

Várias shells

✓ UNIX permite vários programas shell

- o usuário escolhe um de sua preferência

– mais comuns

- sh (Bourne) shell original do UNIX
- csh, tcsh e zsh C shell (Bill Joy)
- ksh, pdksh Korn shell (David Korn)
- bash Bourne Again Shell
muito popular em Linux

existe um grande número de outras shells

características são mais ou menos comuns

TSW

5

bash

– POSIX-compatível

- shell deve estar instalada em **/bin/sh**
 - verificar com comando **ls -l /bin/sh**

– comandos bash

- antes de entrarmos em programação shell é necessário conhecer os comandos básicos da shell

Comandos Linux e MS-DOS guardam alguma
semelhança. Partes do MS-DOS foram inspiradas no
CP/M, que por sua vez foi inspirado no UNIX.

COMMAND.COM shell única do DOS

TSW

6

Comandos UNIX

não tenha medo de experimentar!

- se você **não** possui domínio dos comandos básicos UNIX, está na hora de:
 - explorar os comandos UNIX (Linux)
 - estabelecer uma relação entre os comandos básicos e o MS-DOS
 - familiarizar-se com os conceitos elementares de operação de ambientes UNIX
 - aprender a usar **redirecionamento e pipes**
- exerçete-os listando arquivos diretórios, verificando o estado de processos ...

use man



TSW

7

Resumo de alguns comandos

revisar

cat: lê e mostra arquivos em stout, se não aparece arquivo então stdin, permite concatenação

grep: pesquisa em um ou mais arquivos por linhas que casam com uma expressão regular dada (-l: listar)

ls: lista conteúdo de um diretório dado, se nenhum é dado lista o atual

more: display o conteúdo do arquivo, uma tela de cada vez

chmod: altera as permissões de acesso de um arquivo

touch: verifica se existe arquivo, senão cria

rm: remove arquivo

use man



TSW

8

Redirecionamento

✓ entrada e saída padrão

- **stdin** entrada padrão
- **stdout** saída padrão também **stine** e **stout**
- **entrada e saída padrão e descritores de arquivo**
 - **stdin** - file descriptor 0
 - **stdout** - file descriptor 1
 - **st error out** - file descriptor 2
- **redirecionamento**
 - troca um arquivo padrão por outro arquivo indicado

TSW

9

Redirecionamento (>) de stout

- exemplo simples:

redireciona saída

\$ ls -l > lssaida.txt

salva a saída do comando ls no arquivo lssaida.txt

ls : lista o conteúdo de um diretório dado; se nenhum diretório é dado, lista o diretório atual

- CUIDADO!!!

- o redirecionamento escreve sobre o arquivo (se o arquivo já existe) apagando o conteúdo anterior

TSW

10

Usando cat e >

- comando cat

cat: lê e mostra arquivos em stout; permite concatenação

- comando **cat**

```
$ cat rel.txt > novorel.txt
$ cat rel1 rel2 rel3> novorel.txt
```
- **entrada rápida de texto**

```
$ cat > novorel.txt
exemplo: Ball, B.
Usando Linux, pg 21
>isso é uma linha de texto
>isso é mais uma linha de texto
>[EOF]
```
- para mostrar o resultado

```
$ cat novorel.txt
```

digite Ctrl+D para marca de fim de arquivo



TSW

11

Append >>

✓ >> (append)

- **anexa** a saída ao fim de um arquivo

- evita apagar um arquivo que já existe como em >
- também existe << (raro uso)

TSW

12

Redirecionamento da saída de erro

- usar `>` com o número do descritor de arquivo
 - operador `2>`
 - útil para descartar informação de erro, evitando que apareça na tela
 - exemplo:
 - redirecionamento de stout
 - `$ kill -HUP 1234 > killout.txt 2>killerr.txt`

redirecionamento de saída de erro

Matando um processo de um script: existe o risco do processo já estar morto. Redirecionar as mensagens de erro evita que elas apareçam na tela.

TSW

13

operador `2>`

- `2>` exemplo :

`$ cat rel.txt >novorel.txt`
cat: rel.txt: No such file or directory

mens. erro
vai para
erros.log

`$ cat rel.txt >novorel.txt 2>erros.log`
`$ cat erros.log`
cat: rel.txt: No such file or directory

`$ cat rel.txt >novorel.txt 2>/dev/null`

balde furado universal: `/dev/null`

lab

TSW

14

Redirecionamento (`<`) de stdin

- arquivos de entrada também podem ser redirecionados

- Ex:

exemplo bobo

`$ more < killout.txt`

`more`: mostra na tela o conteúdo do arquivo,
uma tela de cada vez

TSW

15

Pipe

canal

- ✓ processos rodam em pipeline

- saída de um processo é a entrada do outro

- operador `|`:

exemplo: ordenar saída de `ps`
`$ ps > psout.txt` ps: processor status
`$ sort psout.txt > pssort.out`

mais elegante em uma só linha usando `pipe`

`$ ps | sort pssort.out`

pipe com 3 comandos: mostra status de processos ordenados na tela

`$ ps | sort | more`

TSW

16

Expansão de wildcard

curinga

- ✓ revisar:

a Shell expande wildcards

- `*` qualquer caractere (0 ou mais caracteres)
- `?` um único caractere
- `[set]` conjunto específico de caracteres simples
- `[^set]`
- `{ string }` agrupa strings que serão expandidos pela Shell

exemplo:
`$ ls my_{finger,toes}` `$ ls my_fingers`
 `$ ls my_toes`

TSW

17

Shell como linguagem

duas formas

- ✓ escrevendo programas em Shell

forma 1 digitando os comandos e executando-os interativamente

- o prompt normal `$` troca para `>` quando se inicia a digitar comandos na Shell
- a seguir o programa é executado

forma 2 armazenando os comandos em um arquivo e invocando da mesma forma que um programa

- usando um editor de textos comum

scripts

TSW

18

Exemplo de programa interativo

primeira forma

- determinar todos os arquivos que contém o string POSIX

```
$ for arquivo in *
> do
> if grep -l POSIX $arquivo
> then
> more $arquivo
> fi
> done
```

arquivo é uma variável e \$arquivo seu conteúdo
grep e more são comandos

grep: pesquisa em um ou mais arquivos por linhas que casam com uma expressão regular dada (-l: listar)
more: mostra o conteúdo do arquivo, uma tela de cada vez

TSW

19

Programa interativo x script

✓ programa interativo

- desvantagem primeira forma

- digitar o programa cada vez que for necessário
- inibe reuso

✓ shell script

segunda forma

- armazenar programa em um arquivo
- invocar o arquivo

- vantagem

- scripts possibilitam reuso

TSW

20

Exemplo de shell script

✓ arquivo chamado primeiro.sh

- .sh sem significado
- # indica comentário
- #! comentário especial
 - indica qual programa deve ser usado para executar o arquivo
 - /bin/sh =shell default
- exit 0
 - código de retorno
 - 0 indica sucesso

```
#!/bin/sh
# primeiro.sh
# compilar apenas os args.
# contendo string POSIX

for file in *
do
    if grep -l POSIX $file
    then
        more $file
    fi
done
exit 0
```

TSW

21

Executando scripts

- forma fácil

- invocar a shell com o nome do arquivo

\$ /bin/sh primeiro.sh

- forma mais elegante

- invocar diretamente

\$ chmod +x primeiro.sh

chmod: altera as permissões de acesso de um arquivo
+x: adiciona modo executável

- pode não funcionar

- o arquivo pode não ser localizado (se PATH não foi atualizada para procurar no diretório atual)

TSW

22

Variáveis

shell sintax

- não são usualmente declaradas antes de usar

- criadas quando usadas pela primeira vez
- maiúsculas e minúsculas são diferenciadas
- valor de uma variável: \$ antes do nome

```
$ meuscumprimentos=Alo
$ echo $meuscumprimentos
Alo
$ meuscumprimentos= "Tudo bem"
$ echo $meuscumprimentos
Tudo bem
$ meuscumprimentos=9+3
$ echo $meuscumprimentos
9+3
```

precisa de apóstrofes se contém espaços

lab

23

read

- read permite ler do teclado valor de variável

- termina pressionando tecla de return

```
#!/bin/sh
```

```
echo Entre algum texto
```

```
read texto
```

```
echo $texto
```

```
echo '$texto' agora contém $texto
```

mostra conteúdo da variável \$texto

mostra o string \$texto

lab

24

Apóstrofes na shell

- ✓ pequena pausa para falar sobre apóstrofes simples e duplas

mostra string \$myvar
\ remove qualquer significado especial de \$

```
#!/bin/sh
myvar="Hi there"
echo $myvar
echo "$myvar"
echo '$myvar'
echo \$myvar
echo Enter some text
read myvar
echo '$myvar' now equals $myvar
exit 0
```

lab

TSW

25

25

Variáveis de ambiente

- ✓ são variáveis pré-definidas

- valores são obtidos do sistema quando um script inicia execução
 - dependem do ambiente de execução
 - normalmente maiúsculas
 - para distinguir das variáveis do usuário (geralmente minúsculas)
 - dependem da configuração particular do ambiente

exemplos: \$HOME, \$PATH, \${}, \$\$

TSW

26

Algumas variáveis de ambiente

- \$HOME
 - diretório pessoal (*home*) do usuário atual
- \$PATH
 - lista de diretórios onde procurar por comandos
 - separado por : (dois pontos)
- exemplo /usr/local/bin:/bin:/usr/bin:./home/neil/bin
- \${#} número de parâmetros passados
- \$\$ o process ID da shell script

TSW

27

Mais algumas variáveis de ambiente

- \$PS1
 - *command prompt* (usualmente \$)
- \$PS2
 - *command prompt* secundário (usualmente >)
- \$IFS
 - separador de campos de entrada
 - usualmente
 - espaço
 - tab
 - nova linha

TSW

28

Parâmetros

- variáveis adicionais são criadas quando a shell é invocada com parâmetros
 - se a shell for invocada sem parâmetros então \${#} possui o valor 0 (zero)
 - \${1,2,...} parâmetros dados ao script
 - \${*} lista de todos os parâmetros separados pelo 1o. caractere em IFS
 - \${@} variação de \${*} que não usa IFS

IFS: separador de campos de entrada

TSW

29

Exemplo

```
salvar o arquivo com o nome
de tentar_var
#!/bin/sh
salutation="Hello"
echo $salutation
echo "The program $0 is now running"
echo "The second parameter was $2"
echo "The first parameter was $1"
echo "The parameter list was $*"
echo "The user's home dir is $HOME"
echo "Please enter a new greeting"
read salutation
echo $salutation
echo "The script is now complete"
exit 0
```

o que acontece?

executar
\$./tentar_var foo bar baz

lab

TSW

30

Teste de condições

✓ comando de teste

- duas formas equivalentes

```
[ ] não esquecer dos espaços  
test
```

- condições de teste

- 3 categorias
 - comparação de strings
 - comparação aritmética
 - características de arquivos

exemplos serão vistos
junto a estruturas de
controle

TSW

31

Comparação de strings

Comparação

Verdadeiro se:

string1 = string2 São iguais

string1 != string2 Não são iguais

-n string String não é nulo

-z string String é null (string vazia)

TSW

32

Comparação aritmética

Comparação

Verdadeiro se:

expr1 -eq expr2	São iguais
expr1 -ne expr2	Não são iguais
expr1 -gt expr2	expr1 > expr2
expr1 -ge expr2	expr1 >= expr2
expr1 -lt expr2	expr1 < expr2
expr1 -le expr2	expr1 <= expr2
! expr	Not expr

TSW

33

Características de arquivos

verdadeiro se

-d file	É diretório	-s file	Tamanho não é zero
-e file	Existe	-u file	set-user-id está ligado
-f file	É arquivo regular	-w file	Pode ser escrito
-g file	set-group-id está ligado	-x file	É executável
-r file	É legível	-e e -f	são similares -f é mais usada

TSW

34

Estruturas de controle

✓ if

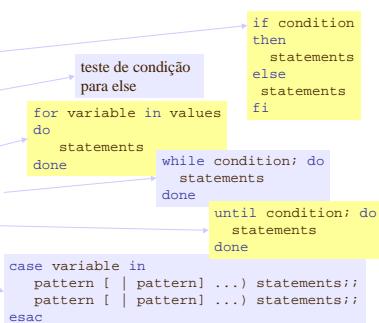
✓ elif

✓ for

✓ while

✓ until

✓ case



TSW

35

if

#!/bin/sh

echo "Is it morning? yes or no"
read timeofday

if [\$timeofday = "yes"]; then

echo "Good morning"

else

echo "Good afternoon"

fi

exit 0

teste: observe espaços

problema se entrar algo
diferente de yes ou no

pode ser resolvido com mais
um teste no ramo else

lab

36

elif

```
#!/bin/sh
echo "Is it morning? Please answer yes or no"
read timeofday

if [ $timeofday = "yes" ]
then
    echo "Good morning"
    elif [ $timeofday = "no" ]; then
        echo "Good afternoon"
    else
        echo "Sorry, $timeofday not recognized. Enter yes or no"
    exit 1
fi
script termina com valor 1

exit 0
```

ainda tem um problema se entra `return`

lab

TSW

37

problema???

entrando com `return`

- problema com o if
 - `timeofday` será considerado como:

`if ["yes"]` condição inválida
gera mensagem de erro

- para evitar esse problema

- usar apóstrofes

`if ['$timeofday' = "yes"]`

- `timeofday` será considerado como:

`if ["" = "yes"]` condição válida

TSW

38

elif & if & teste com apóstrofes

```
#!/bin/sh
echo "Is it morning? Please answer yes or no"
read timeofday
if [ "$timeofday" = "yes" ]
then
    echo "Good morning"
elif [ "$timeofday" = "no" ]; then
    echo "Good afternoon"
else
    echo "Sorry, $timeofday not recognized."
    echo "Enter yes or no"
    exit 1
fi
exit 0
```

lab

TSW

39

for

```
for variable in values
do
    statements
done
```

`for foo in bar fud 43`

lista de valores bar fud 43

`$(comand)`

```
#!/bin/sh
for file in $(ls f*.sh); do
    echo $file
done
exit 0
```

lab

40

while

```
#!/bin/sh
while condition; do
    statements
done
```

`echo "Enter password"`

`read trythis`

`while ["$trythis" != "secret"]; do`

`echo "Sorry, try again"`

`read trythis`

`done`

`exit 0`

`while condition; do`

`statements`

`done`

apóstrofes desnecessárias

`$(())`

```
#!/bin/sh
foo=1
while [ "$foo" -le 20 ]
do
    echo "Here we go again"
    foo=$((foo+1))
done
exit 0
```

lab

TSW

41

until

```
until condition; do
    statements
done
```

`#!/bin/sh`

`until who | grep "$1" > /dev/null`

`do`

`sleep 60`

`done`

`loop até usuário com login dado pelo parâmetro $1 logar-se na máquina`

`# Now ring the bell`

`# and announce the expected user`

`echo -e \\a` toca o alarme

`echo *** $1 has just logged in ***`

`exit 0`

TSW

42

case

```
#!/bin/sh
echo "Is it morning? Please answer yes or no"
read timeofday
executa o primeiro comando que casa e finaliza
case "$timeofday" in
    "yes") echo "Good Morning";;
    "no" ) echo "Good Afternoon";;
    "y"  ) echo "Good Morning";;
    "n"  ) echo "Good Afternoon";;
    *   ) echo "Sorry, answer not recognised";;
esac
exit 0
```

bom aparecer sempre no fim

TSW

43

case

```
#!/bin/sh
echo "Is it morning? Please answer yes or no"
read timeofday
case "$timeofday" in
    yes | y | Yes | YES ) echo "Good Morning";;
    n* | N* | * ) echo "Good Afternoon";;
esac
exit 0
```

exemplo do uso de * em case

expansão do wildcard * em N* e n* pode casar com um grande número de palavras, não apenas NO e no

TSW

44

case

outra variação

```
#!/bin/sh
echo "Is it morning? Please answer yes or no"
read timeofday
case "$timeofday" in
    yes | y | Yes | YES )
        echo "Good Morning"
        echo "Up bright and early this morning?"
        ;;
    [nN]* )
        echo "Good Afternoon"
        ;;
    * )
        echo "Sorry, answer not recognised"
        echo "Please answer yes or no"
        exit 1
        ;;
esac
exit 0
```

saida com código 1

TSW

45

Listas de comandos

✓ permite executar uma série de comandos

- lista E (*AND list*)

- executa o próximo comando apenas se todos os anteriores executam **com** sucesso
- executa enquanto comando retorna TRUE

- lista OU (*OR list*)

- executa o próximo comando apenas se todos os anteriores executam **sem** sucesso
- executa enquanto comando retorna FALSE

TSW

46

Lista E

```
statement1 && statement2 && statement3 && ..
```

```
#!/bin/sh
touch f_one
rm -f f_two
if [ -f f_one ] && echo "hello" && [ -f f_two ] && echo "there"
then
    echo -e "in if"
else
    echo -e "in else"
fi
exit 0
```

teste retorna FALSE

não executa

ramo else é executado porque condition é FALSE (terceiro comando da lista AND)

touch : verifica se arquivo existe, senão cria

rm : remove arquivo

TSW

47

Lista OU

```
statement1 || statement2 || statement3 || ..
```

```
#!/bin/sh
rm -f f_one
if [ -f f_one ] || echo "hello" || echo "there"
then
    echo -e "in if"
else
    echo -e "in else"
fi
exit 0
```

teste retorna FALSE

executa

não executa

ramo then é executado porque condition é TRUE (segundo comando da lista OR)

rm : remove arquivo

TSW

48

Listas E e OU

- listas E e OU retornam valor do último comando avaliado
 - lista E: geralmente **falso** (a menos que todos os comandos da lista retornem verdadeiro)
 - lista OU: geralmente **verdadeiro** (a menos que todos os comandos da lista retornem falso)
- listas E e OU podem ser combinadas
 - exercício para especialistas em lógica

exemplo: [-f .profile] && exit 0 || exit 1

TSW

49

Blocos de comandos

✓ usar { }

- permitem construir um bloco de comandos
- blocos são usados para colocar múltiplos comandos onde apenas um comando é permitido

```
{  
statement  
statement  
statement  
}
```

TSW

50

Funções

- ✓ facilitam a escrita de programas grandes
 - mesma argumentação usada para linguagens de programação
 - alternativa seria chamar programas dentro de programas
 - funções são mais rápidas e a passagem de parâmetros é mais simples

✓ definição

```
function-name () {  
statements  
}
```

TSW

51

Função: exemplo

```
#!/bin/sh  
  
foo() {  
    echo "Function foo is executing"  
}  
  
echo "script starting"  
foo _____  
echo "script ended"  
  
exit 0
```

lab

52

Parâmetros

\$1, \$2, ... parâmetros dados ao script
\$* lista de todos os parâmetros

✓ passagem de parâmetros

- quando uma função é invocada, os **parâmetros posicionais** são substituídos pelos parâmetros da função
- quando a função termina, os **parâmetros posicionais** são restaurados ao valor inicial

✓ comando return

- forma da função retornar valores
 - retorna valores **numéricos**

TSW

53

Função: mais um exemplo ...

fonte: my_name

```
#!/bin/sh  
  
definição da função  
  
yes_or_no() {  
    echo "Is your name $* ?"  
    while true  
    do  
        echo -n "Enter yes or no: "  
        read x  
        case "$x" in  
            y | yes ) return 0;;  
            n | no ) return 1;;  
            * ) echo "Answer yes or no"  
        esac  
    done  
}  
  
retorno de valor numérico  
0 equivale a verdadeiro  
1 equivale a falso
```

lab

54

Função: mais um exemplo

```

        fonte: my_name
echo "Original parameters are $*" lista de todos parâmetros
if yes_or_no "$1" chamada de função com
then passagem de um parâmetro
  echo "Hi $1, nice name"
else
  echo "Never mind"
fi teste de valor numérico
exit 0 0 verdadeiro
        1 falso
    
```

programa principal com invocação da função

executar com `./my_name.sh Taisy Weber`

TSW

55

lab

Comandos

✓ normais

qualquer comando UNIX válido

- comandos que podem ser executados a partir do prompt da shell, não apenas dentro de um script

✓ internos (*built-in*)

- comandos que só podem ser executados dentro de um script
- não podem ser invocados externamente
 - para alguns deles existem programas externos correspondentes
 - os internos são mais eficientes

TSW

56

Alguns comandos

✓ break	✓ export
✓ :	✓ expr ver \$(())
✓ continue	✓ printf gera saída formatada, só disponível recentemente
✓ . / mostra um string	✓ return
✓ echo	✓ set
✓ eval	✓ shift
✓ exec	✓ trap
✓ exit n	✓ unset

TSW

57

Execução de comandos

✓ \$(command)

evitar forma antiga 'command'

- captura o resultado da execução de um comando e permite colocar em uma variável
 - resultado é uma *string* (saída do comando)
 - resultado não é estado de retorno

```

#!/bin/sh
echo diretório atual é $PSW
echo usuário atual é $wo
exit 0
  
```

PSW variável de ambiente (não comando)

wo é comando e por isso leva ()

TSW

58

Execução de comandos: expansão aritmética

✓ \$((. . .))

- alternativa ao comando expr

- expr é lento pois invoca nova shell para execução da expressão

```

#!/bin/sh

x=0
while [ "$x" -ne 10 ]; do
  echo $x
  x=$((x+1))
done

exit 0
  
```

TSW

59

expressões

usadas em expr e em \$((...))

Expressão	Descrição
expr1 expr2	expr1 se expr1 não é zero, senão expr2
expr1 & expr2	zero se qualquer expr for zero, senão expr1
expr1 = expr2	igual
expr1 > expr2	maior que
expr1 <= expr2	maior que ou igual
expr1 < expr2	menor que
expr1 <= expr2	menor que ou igual
expr1 != expr2	diferente
expr1 + expr2	soma
expr1 - expr2	subtração
expr1 * expr2	multiplicação
expr1 / expr2	divisão inteira
expr1 % expr2	módulo

} aritméticas

TSW

60

Fim de programação usando shell

- vimos uma breve introdução ao potencial de programação usando shell
- domínio dos recursos vem com a prática
 - muitos dos problemas de administração de um sistema UNIX podem ser facilmente resolvidos com programas shell aproveitando os recursos disponíveis