Shell Script

Arthur, Luis Filipe, Rodrigo

Linguagem Script

Linguagem de script (também conhecido como linguagem de scripting, ou linguagem de extensão) são linguagens de programação executadas do interior de programas e/ou de outras linguagens de programação, não se restringindo a esses ambientes. As linguagens de script servem para estender a funcionalidade de um programa e/ou controlá-lo, acessando sua API e, são frequentemente usadas como ferramentas de configuração e instalação em sistemas operacionais (Shell script), como por exemplo, em alguns sistemas operacionais da família Linux, que usam a linguagem bash.

Introdução

O primeiro shell do Unix foi o Thompson shell, criado por Ken Thompson nos laboratórios da AT&T Bell em New Jersey, EUA. Distribuído entre as versões de 1 a 6 do Linux entre 1971 a 1975. Considerado bem rudimentar para padrões modernos. Depois aperfeiçoado por John Mashey e outros e distribuído de 1975 a 1977.

Bash

Bash é o shell, ou interpretador de comandos da linguagem, do sistema operacional GNU. O nome é um acrônimo de “bourne-again shell” , uma piada com o nome de Stephen Bourne, autor do ancestral direto do Unix Shell atual.

O que é o Shell?

O Unix Shell é ao mesmo tempo um interpretador de comandos e uma linguagem de programação. Como interpretador de comandos, ele dá acesso ao rico conjunto de utilidades do GNU e como linguagem de programação ele permite que tais utilidades sejam combinadas. Arquivos contendo comandos podem ser criados e se tornar comandos. Esses novos comandos tem o mesmo status de comandos de sistema como os do diretório /bin.

Quando Não Usar

When not to use shell scripts

- Resource-intensive tasks, especially where speed is a factor (sorting, hashing, recursion [2] ...)
- Procedures involving heavy-duty math operations, especially floating point arithmetic, arbitrary precision calculations, or complex numbers (use *C++* or *FORTRAN* instead)
- Cross-platform portability required (use *C* or *Java* instead)
- Complex applications, where structured programming is a necessity (type-checking of variables, function prototypes, etc.)
- Mission-critical applications upon which you are betting the future of the company
- Situations where *security* is important, where you need to guarantee the integrity of your system and protect against intrusion, cracking, and vandalism
- Project consists of subcomponents with interlocking dependencies
- Extensive file operations required (*Bash* is limited to serial file access, and that only in a particularly clumsy and inefficient line-by-line fashion.)
- Need native support for multi-dimensional arrays
- Need data structures, such as linked lists or trees
- Need to generate / manipulate graphics or GUIs
- Need direct access to system hardware or external peripherals
- Need port or socket I/O
- Need to use libraries or interface with legacy code
- Proprietary, closed-source applications (Shell scripts put the source code right out in the open for all the world to see.)

Shell é compilado ou interpretado?

Shell é uma linguagem interpretada.

Obs: shc - Generic shell script compiler (http://www.datsi.fi.upm.es/~frosal/)

Palavras Reservadas

!:	Pipelines
[ ]:	Conditional Constructs
{ }:	Command Grouping
case:	Conditional Constructs
do:	Looping Constructs
done:	Looping Constructs
elif:	Conditional Constructs
else:	Conditional Constructs
esac:	Conditional Constructs
fi:	Conditional Constructs
for:	Looping Constructs
function:	Shell Functions
if:	Conditional Constructs
in:	Conditional Constructs
select:	Conditional Constructs
then:	Conditional Constructs
time:	Pipelines
until:	Looping Constructs
while:	Looping Constructs

Tipos de Dados

- Tipagem Fraca

- Tipagem Dinâmica

Tipos de Dados

var=10

var=“Sou uma string”

var=25,12

var[0]=a

var[5]=12

Obs: echo ${var[0]}

Tipos de Dados

echo ${var[@]}
echo ${var[*]}
imprime todos os conteúdos

echo ${!var[@]}
echo ${!var[*]}
imprime todos os índices

Variáveis e Constantes

-Variáveis devem ser declaradas junto com seu valor.

-Todas as variáveis são globais por definição a não ser que os comandos “local” ou “declare” sejam usados.

Variáveis e Constantes

Para acessar o conteúdo de uma variável utilizamos o símbolo “$”.

Exemplo:

```
var=10
echo $var
```

Variáveis e Constantes

Constantes podem ser criadas utilizando-se o comando “readonly”.

Exemplo:

readonly constante=abc

constante=def (não irá aceitar por ser read-only)

Variáveis e Constantes

Existe uma maneira de se especificar o tipo de uma variável, utilizando-se o comando “declare”.

Exemplo:

declare -i variable=12

variable=“macaco” (variable fica com valor 0)

Variáveis e Constantes

Este comando também serve para restringir o escopo de uma variável:

Exemplo:

```
foo () {
        FOO="bar"
}
bar () {
        foo
        echo $FOO
}
bar    # Imprime bar.
```

Porém . . .

```
foo ()  {
        declare FOO="bar"
}
bar () {
        foo
        echo $FOO
}
bar  # Imprime nada.
```

Variáveis e Constantes

Opções do comando declare:

-a Variável é um vetor.

-f Lista todas as funções declaradas.

-i Variável é um inteiro.

-p Mostra os atributos e valores de cada variável.

-r Faz com que as variáveis sejam read-only (constantes).

Trocando o - por + podemos remover um atributo da variável.

Variáveis e Constantes

O valor pode ser expressado entre as aspas (“”), apóstrofos (‘’) ou crases (``).

```
variavel="Eu estou logado como usuário $user"
echo $variavel
Eu estou logado como usuário cla

variavel='Eu estou logado como usuário $user'
echo $variavel
Eu estou logado como usuário $user

variavel="Meu diretório atual é o `pwd`"
echo $variavel
Meu diretório atual é o /home/cla
```

Gerenciamento de memória

As variáveis são criadas como variáveis de ambiente, deixando o sistema operacional responsável pelas gerência da memória.

Operadores

Operadores Aritméticos	
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo
**	Exponenciação

Operadores

Operadores de Atribuição	
=	Atribui valor a uma variável
+=	Incrementa a variável por uma constante
-=	Decrementa a variável por uma constante
*=	Multiplica a variável por uma constante
/=	Divide a variável por uma constante
%=	Resto da divisão por uma constante
++	Incrementa em 1 o valor da variável
--	Decrementa em 1 o valor da variável

Operadores

Operadores Aritméticos

```
a=10

a=$(expr $a + 1)

a=$(expr $a % 3)

a=$(expr $a \* 10)

a=$((a**2))

a=$((a*2))
```

Operadores de Atribuição

```
b=15

((b+=5))

let b+=1

let b=b/5

let b*=2

let b--
```

Obs.:

b+=4 => ~ $ 154

Operadores de BIT

Operadores de BIT	
<<	Deslocamento à esquerda
>>	Deslocamento à direita
&	E de bit (AND)
|	OU de bit (OR)
^	OU exclusivo de bit (XOR)
~	Negação de bit
!	NÃO de bit (NOT)
Operadores de BIT (atribuição)	
<<=	Deslocamento à esquerda
>>=	Deslocamento à direita
&=	E de bit
|=	OU de bit
^=	OU exclusivo de bit

Operadores

```
a=3
b=123

echo " \$a & \$b é = $(($a & $b))"
echo " \$a | \$b é = $(($a | $b))"
echo " \$a ^ \$b é = $(($a ^ $b))"

=>  $a & $b é = 3
=>  $a | $b é = 123
=>  $a ^ $b é = 120
```

```
c=2          binário = 0000 0010
c=$(( c << 2 ))

echo $c
=>  8        binário = 0000 1000

d=13
(( d&=3 ))

=>   1
```

String

var1=“Sou uma string”
var2=“em Shell Script”
var3=“$var1 $var2”

$var3 == “Sou uma string em Shell Script”

Alterando uma string para um vetor

vetor=(${var3})

vetor[0]=Sou
vetor[1]=uma
vetor[2]=string

${vetor[@]} == “Sou uma string”

Alterando um vetor para uma string

var=${vetor[#]}

Comparadores

Comparação Numérica	
`-lt`	É menor que (LessThan)
`-gt`	É maior que (GreaterThan)
`-le`	É menor igual (LessEqual)
`-ge`	É maior igual (GreaterEqual)
`-eq`	É igual (EQual)
`-ne`	É diferente (NotEqual)
Comparação de Strings	
`=`	É igual
`!=`	É diferente
`-n`	É não nula
`-z`	É nula

Controle de fluxo

Operações condicionais

```
if [ $num -lt 5 -o $num -gt 15 ]
then
        {...}
elif (( $num <= 10 ))
then
        {...}
else
        {...}
fi
```

```
if [ $string == “string” ] && (( $num == 1 ))
then
        {...}
elif [ -n string ]
then
        {...}
else
        {...}
fi
```

Comandos de repetição

Operações de repetição

```
for ((i=1;i<=10;i++))
do
    {…}
done
for i in “for” “com” “string”
do
    {…}
done
for i in `seq 1 10`
```

```
while :
do
case $VAR in
    txt1) ... ;;
    txt2) ... ;;
    txtN) ... ;;
    *)    ... ;;
esac
done
```

Leitura/Escrita de Arquivo

Leitura

```
while read line
do
	for i in `seq 1 2`
	do
	var[i]=`echo $line | cut -d';' -
f"$i"`
	done
done < arquivo
```

Escrita

```
echo $var1 $var2 > arquivo

echo $var3 $var4 >> arquivo2
```

Blocos e Agrupamentos

{...}	Agrupa comandos em um bloco
(...)	Executa comandos numa subshell
$(...)	Executa comandos numa subshell, retornando o resultado
((...))	Testa uma operação aritmética, retornando 0 ou 1
$((...))	Retorna o resultado de uma operação aritmética
[...]	Testa uma expressão, retornando 0 ou 1 (alias do comando 'test')
[[...]]	Testa uma expressão, retornando 0 ou 1 (podendo usar && e ||)

Expressão regular

Uma expressão regular provê uma forma concisa e flexível de identificar cadeias de caracteres de interesse, como caracteres particulares, palavras ou padrões de caracteres.

~$ grep [0-9]* arquivo

Comandos Internos

alias: Bash Builtins
bg: Job Control Builtins
bind: Bash Builtins
break: Bourne Shell Builtins
builtin: Bash Builtins
caller: Bash Builtins
cd: Bourne Shell Builtins
command: Bash Builtins
compgen: Programmable Completion Builtins
complete: Programmable Completion Builtins
compopt: Programmable Completion Builtins
continue: Bourne Shell Builtins
declare: Bash Builtins
dirs: Directory Stack Builtins
disown: Job Control Builtins
echo: Bash Builtins
enable: Bash Builtins
eval: Bourne Shell Builtins
exec: Bourne Shell Builtins
exit: Bourne Shell Builtins
export: Bourne Shell Builtins
fc: Bash History Builtins
fg: Job Control Builtins
getopts: Bourne Shell Builtins

Comandos Internos

hash: Bourne Shell Builtins
help: Bash Builtins
history: Bash History Builtins
jobs: Job Control Builtins
kill: Job Control Builtins
let: Bash Builtins
local: Bash Builtins
logout: Bash Builtins
mapfile: Bash Builtins
popd: Directory Stack Builtins
printf: Bash Builtins
pushd: Directory Stack Builtins
pwd: Bourne Shell Builtins
read: Bash Builtins
readarray: Bash Builtins
readonly: Bourne Shell Builtins
return: Bourne Shell Builtins
set: The Set Builtin
shift: Bourne Shell Builtins
shopt: The Shopt Builtin
source: Bash Builtins
suspend: Job Control Builtins
test: Bourne Shell Builtins
times: Bourne Shell Builtins
trap: Bourne Shell Builtins
type: Bash Builtins

Comandos Internos

typeset: Bash Builtins
ulimit: Bash Builtins
umask: Bourne Shell Builtins
unalias: Bash Builtins
unset: Bourne Shell Builtins
wait: Job Control Builtins

Comandos básicos

Comando	Descrição	Sintaxe
echo	Exibe o texto na tela	echo “texto a ser mostrado”
sleep	Dá um tempo antes de seguir	sleep segundos **exemplo:** Sleep 1
read	Recebe o valor de uma variável (próxima aula)	read variável **exemplo:** read dados
>	Escreve num arquivo texto *(apagando o que estava lá)*	echo “texto” > /home/luiz/arquivo
>>	Escreve num arquivo texto *(na ultima linha)*	echo “texto” >> /home/luiz/arquivo
&	Roda o comando em 2º plano e continua o script	Comando&
exit	Sai do script	exit
touch	Cria arquivos texto	touch arquivo
#	Comenta tudo depois deste simbolo	# Comentário

Comandos básicos

Diretórios		
Comando	**Sintaxe**	**Descrição**
rm -rf	rm -rf +diretório	Deleta arquivos/pastas e tudo que estiver dentro (cuidado)
pwd	pwd	Mostra em qual diretório estamos
chmod	chmod 777 arquivo_ou_pasta	Muda as permissões, 777 = permissão total
chown	chown user:grupo arq_ou_diret.	Muda o proprietário de arquivos e pastas
cd	cd diretório	Entra em diretórios

Comandos básicos

Usuários		
Comando	**Sintaxe**	**Descrição**
useradd	useradd luiz -g alunos (no grupo)	Adiciona um usuário
userdel	userdel usuário	Deleta usuário e seus arquivos
groupdel	groupdel grupo	Deleta um grupo
groups	groups nome_usuario	Mostra os grupos do usuário
addgroup	addgroup usuario grupo ou addgroup nomedogrupo	Cria um grupo ou adiciona um usuário ao grupo
sudo	sudo comando	Executa comandos como root
whoami	whoami	Identifica com qual usuário você esta logado

Comandos básicos

Rede		
Comando	**Sintaxe**	**Descrição**
ifconfig	ifconfig	Mostra as interfaces de rede
hostname	hostname	Mostra ou muda o nome de seu computador na rede
ping	Ping ip_desejado	Dispara pacotes para outro pc, para testar conexões etc

Comandos básicos

Sistema		
Comando	**Sintaxe**	**Descrição**
killall	Killall nome_do_programa	Mata um processo
whatis	Whatis +nome do programa	Descreve o que faz o comando
diff	diff arquivo1 arquivo2	Compara 2 arquivos
ps	ps -elf	Mostra os programas que estão rodando
cat	cat arquivo_texto	Mostra o conteúdo de um arquivo de texto
grep	Comando | grep palavra	Filtra a saída do comando, mostra a linha da palavra pedida
ln	ln -s arquivo_original atalho	Cria atalho
cp	cp arquivo destino	Copia um arquivo ou diretório (-R para diretórios)
apt-get	apt-get nome_programa	Instala aplicativos
find	Find +nome	Procura por arquivos e diretórios

Funções

```
imprime ()
{
        echo "Sou o programa $0"
        echo "Recebi $# parametros"
        echo "Param 1: $1"
        echo "Param 2: $2"
        echo "Lista de parâmetros: $*"
}
imprime um dois tres quatro
echo "Sou o programa $0"
echo "Recebi $# parametros"
echo $1 $2 $3
```

```
$ ./teste.sh a b c

Sou o programa teste.sh
Recebi 4 parametros
Param 1: um
Param 2: dois
Lista de parâmetros: um dois tres quatro
Sou o programa teste.sh
Recebi 3 parametros
a b c
```

Funções

```
function retorna() {
    echo "sou um valor"
    return 42
}
valor=$(retorna)

echo $?
echo $valor
```

```
~$ 42
~$ sou um valor
```

Usando vários scripts em um só

Pode-se precisar criar vários scripts shell que fazem funções diferentes, você só precisa incluir o seguinte comando no seu script shell:

. script2.sh

ou

source script2.sh

Orientação à Objetos

Não possui.

Polimorfismo

Não possui.

Tratamento de Exceções

Não possui.

Tipos abstratos de dados

Não possui.

Avaliação da Linguagem

- Facilidade de aprendizado
- Podemos criar novos comandos com facilidade
- Desenvolvimento rápido
- Facilidade de desenvolvimento e manutenção

Avaliação da Linguagem

- Não é eficiente
- Baixa legibilidade
- Baixa confiabilidade
- Não oferece vários recursos comuns em outras linguagens

Referências

http://www.gnu.org/software/bash/manual/bash.html

http://tldp.org/LDP/Bash-Beginners-Guide/html/

http://shellscript.com.br

http://tldp.org/LDP/abs/html/index.html

Jargas, A. M. Shell Script Profissional