Programação para Servidores

Professor Marcos Monteiro, MBA, ITIL

Objetivos gerais

- Criar scripts para personalização de servidores e execução automática de tarefas administrativas.

Conteúdos

Indicação do material didático

- Título: Classic Shell Scripting
  - Autor: Arnold Robbins
  - Editora: Artmed
  - Ano: 2008
  - Capítulos:
    - 2 (20 páginas)
    - 3 (32 páginas)
    - 5 (20 páginas)
    - 6 (28 páginas)
    - 7 (34 páginas)
    - 15 (9 páginas)
    - Apêndice C (5 páginas)

Total de 148 páginas

Aula 01

Introdução

Definição de shell script

- Shell script é uma linguagem de programação interpretada usada em vários sistemas operacionais.

- Na linha de comandos de um interpretador de comandos (shell) pode-se utilizar diversos comandos um após o outro, ou mesmo combiná-los numa mesma linha. Se forem colocados diversas linhas de comandos em um arquivo texto simples, tem-se em um shell script. Uma vez criado, um shell script pode ser reutilizado quantas vezes for necessário.

- Todo sistema Unix e similares são repletos de scripts em shell para a realização das mais diversas atividades administrativas e de manutenção do sistema. Os arquivos de lote (batch) do Windows são também exemplos de shell scripts.

- Por serem facilmente agendados para execução através do crontab, os shell scripts são usados para construções de ferramentas indispensáveis aos administradores de sistemas Unix.

- Dentre as principais razões para se utilizar shell scripts, podem ser citadas:

- Simplicidade ? Por ser uma linguagem de alto nível, é possível expressar operações complexas de forma e simples.

- Portabilidade ? Por ser universal entre sistemas Unix, existe uma grande chace de um shell script escrito para um sistema ser transferido para outro sem necessidade de alterações.

- Facilidade de desenvolvimento ? Pode-se desenvolver um shell script poderoso e útil em pouco tempo.

Programação script

- O conhecimento básico de programação script é essencial para quem deseja tornar-se um administrador de sistemas. Durante o processo de boot uma máquina Linux executa os shell scripts em /etc/rc.d para configurar o sistema e os serviços. Uma compreensão detalhada de tais scripts de inicialização é importante para analisar o comportamento de um sistema e, possivelmente, modificá-lo.

- A programação script não é difícil de dominar, pois scripts podem ser construídos em pequenas seções, existindo apenas um conjunto relativamente pequeno de operadores específicos e opções para aprender. A sintaxe é simples e direta, semelhante à execução de forma encadeada de utilitários na linha de comando, e há apenas algumas regras que regem a sua utilização.

Scripts versus linguagens compiladas

- A maioria dos programas, notadamente os comerciais, são escritos em linguagens compiladas. Tais programas devem ser traduzidos para um código objeto (por intermédio de um compilador) e posteriormente ligados a funções em bibliotecas (por intermédio de um linkeditor) a fim de se obter um programa executável. Os programas executáveis podem ser executados diretamente pelo hardware do computador. A grande vantagem na utilização programas escritos em linguagens compiladas é a performance que se obtém.

- Linguagens script geralmente são linguagens interpretadas, ou seja, as instruções são buscadas no programa uma após outra a fim de serem executadas. A grande vantagem na utilização de scripts é sua simplicidade e rapidez para o desenvolvimento de tarefas simples e repetitivas, características da administração de sistemas.

#

- Linhas que se iniciam com o caractere # são consideradas comentário, ou seja não são executadas. Também são comentários caracteres que venha logo após um # (desde que haja pelo menos um caractere em branco antes do #).

;

- Separador de comandos. Utilizado para permitir a execução de dois ou mais comandos em uma mesma linha.

;;

- Terminador do comando case.

.

- O caractere ponto representa um caractere qualquer em uma expressão regular.

"

- Protege de interpretação vários caracteres especiais da string que se encontrar entre aspas.

'

Protege de interpretação todos os caracteres especiais da string que se encontrar entre aspas simples.

\

Funciona como escape para permitir que caracteres especiais tenham seu significado literal.

/

Separador de nomes em um caminho de diretório.

`

Substituição de comando.

?

Operador de teste.

$

Substituição de variável ou padrão.

( )

Agrupamento de comandos.

PERGUNTAAA!

Qual a diferença entre linguagens script e linguagens compiladas?

Qual a correlação entre scripts e interpretadores de comandos?

Estudo dirigido

leitura do capítulo 2 do livro “Classic Shell Scripting”.

Aula 02
Variáveis e Testes

Objetivo:

Entender como funciona a substituição de variáveis em shell script.

Conhecer os tipos de variáveis especiais utilizados em programação shell.

Conhecer as formas de proteção de variáveis com aspas.

Conhecer os mecanismos de testes em shell script.

Variáveis

Variável é a forma que as linguagens de programação utilizam para representar dados. Uma variável nada mais é do que um rótulo, um nome atribuído a uma posição ou conjunto de posições na memória do computador contendo um item de dados. Cada variável possui um valor, que representa a informação que foi atribuída a ela.

Em shell script, o nome de uma variável inicia sempre por uma letra ou por um sublinhado, e pode conter qualquer quantidade de letras, dígitos e/ou sublinhados. O conteúdo de uma variável é armazenado como um string e não há limite para a quantidade de caracteres que ela pode conter.

A atribuição de valores a uma variável é feita escrevendo-se o nome da variável seguido imediatamente do caractere = e o valor a ser atribuído, sem qualquer espaço entre eles. Para atribuir a uma variável valor que contenha espaços, o valor deverá ser colocado entre aspas. Alguns exemplos de atribuição de valores a variáveis:

tipo=jogador

nome=”Edson Nascimento”

Variáveis

Para obter o valor de uma variável utiliza-se o caractere $ precedendo seu nome. Por exemplo:

```
echo $nome
```

Para limpar o valor de uma variável basta fazer uma atribuição nula. Por exemplo:

```
nada=
```

Ao contrário de muitas linguagens de programação, o shell não separa suas variáveis por tipo. Em shell script as variáveis são sequências de caracteres (strings), mas dependendo do contexto podem ser realizadas operações aritméticas sobre as variáveis.

Argumentos

Argumentos são passados para o script a partir da linha de comando por intermédio das variáveis $0, $1, $2, $3, ... , onde $0 é o nome do próprio script, $1 é o primeiro argumento, $2 o segundo argumento, e assim por diante. Depois de 9, os argumentos devem ser colocados entre parênteses. Por exemplo, $(10), $(11), $(12), ...

Proteção com aspas

Proteção com aspas (quoting) é a forma utilizada para informar ao shell como interpretar os dados passados.

Escape com barra invertida

Preceder um caractere com uma barra invertida (\) diz ao shell para tratar literalmente o caractere.

Aspas simples

Aspas simples ('.....') forçam o shell a tratar literalmente tudo o que estiver entre o par de aspas.

Não é possível encaixar aspas simples dentro de uma string protegida por aspas simples, pois nem a barra invertida é especial dentro de aspas simples.

Aspas duplas

Aspas duplas (".....") agrupam o texto como uma única string, porém o shell processa caracteres de escape, variáveis, substituição de comandos, etc.

Testes

O construtor if/then testa se a saída do comando anterior é 0 (em UNIX, 0 significa sucesso), e em caso afirmativo, executa uma ou mais comandos.

Existe um comando específico, denominado [(caractere especial colchete esquerdo). Ele considera seus argumentos como expressões de comparação ou testes em arquivos e retorna um status de saída correspondente ao resultado da comparação (0 para verdadeiro e 1 para falso).

Em versões mais recentes do shell foi introduzido o comando [[...]], que estendeu os comandos de teste realizando comparações de uma forma mais familiar para os programadores de outras linguagens.

O construtor if/then pode testar a saída de qualquer comando, não se limitando apenas a testes entre colchetes.

Teste 1

```
if [ 1 ]
then
  echo "verdadeiro"
else
  echo "falso"
fi
```

Teste 2 - amplamente

```
if [ condicao1 ]
then
   commando1
   commando2
   commando3
elif [ condicao2 ]
then
   command4
   command5
else
   comando_default
fi
```

Teste em arquivos

-e
  arquivo existe
-f
  é um arquivo regular (não é diretório nem dispositivo)
-s
  não tem tamanho zero
-d
  é um diretório
-b
  é um dispositivo de bloco

Teste em arquivos

-c

é um dispositivo de caractere

-r

tem permissão de leitura

-w

tem permissão de escrita

-x

tem permissão de execução

```
if [ -f $arquivo ]
then
  echo "$arquivo e um arquivo regular"
else
  echo "$arquivo nao existe ou nao e um arquivo
   regular"
fi
```

Mais detalhes sobre testes em arquivos podem ser encontrados no capítulo 7.2 do livro Advanced Bash-Scripting Guide, de M. Cooper, cuja revisão 6.2 em formato eletrônico encontra-se anexada ao plano de ensino desta matéria.

Operadores de comparação
Comparação de inteiros:

-eq
igual

-ne
diferente

-gt
maior que

-ge
maior ou igual

-lt
menor que

-le
menor ou igual

Exemplo

```
if [ "$a" -ne "$b" ]
then
  echo "$a nao e igual a $b"
fi
```

Comparação de strings:

=

igual

==

igual (mesmo que =)

!=

diferente

<

menor que (em ordem alfabética ASCII)

>

maior que (em ordem alfabética ASCII)

-z

string é nula (tem tamanho zero)

-n

string não é nula

Exemplo

```
if [ -n "$str" ]
then
  echo "a string nao e nula"
else
  echo "a string e nula"
fi
```

Comparação composta:

-a AND - “e” lógico
-o OR - “ou” lógico

```
if [ "$expr1" -a "$expr2" ]
then
  echo "expr1 e expr2 sao verdadeiras"
else
  echo "expr1 ou expr2 eh falsa"
fi
```

Oque faz isto??

```
if [ -f $HOME/.Xclients ]; then
  exec $HOME/.Xclients
elif [ -f /etc/X11/xinit/Xclients ]; then
  exec /etc/X11/xinit/Xclients
else
    # failsafe settings.  Although we should never get here
    # (we provide fallbacks in Xclients as well) it can't hurt.
    xclock -geometry 100x100-5+5 &
    xterm -geometry 80x50-50+150 &
    if [ -f /usr/bin/netscape -a -f /usr/share/doc/HTML/index.html ]; then
        netscape /usr/share/doc/HTML/index.html &
    fi
fi
```

Estudo dirigido

Leitura do capítulo 6 do
livro “Classic Shell Scripting”.

Aula 03
Operadores, repetições e tomadas de decisões

Conhecer os operadores utilizados em shell script

Compreender as estruturas de repetições e desvios

Discernir qual a melhor estrutura a ser utilizada em cada caso

Ser capaz de utilizar estruturas de repetição e tomadas de decisão na construção de scripts

Operadores

O principal operador em shell scrip é o operador de atribuição (=). Ele atribui um valor a uma variável, porém o faz interpretando tudo o que vem a direita do = como uma string. Por exemplo, o comando

a=5+3

armazena a string “5+3” na variável a. Para efetuar a operação matemática deve ser utilizado o comando let:

let a=5+3

que neste caso irá efetuar a operação de soma e armazenar o resultado “8” na variável a.

Os interpretadores de comandos mais atuais trabalham com variáveis inteiras de 64 bits, enquanto números em ponto flutuante são tratados como strings.

Operadores matemáticos:

+	soma
-	subtração
*	multiplicação
/	divisão
**	exponenciação
%	módulo

+=	incremento por uma constante
-=	decremento por uma constante
*=	multiplica por uma constante
/=	divide por uma constante
%=	módulo por uma constante

Operadores bit a bit:

<< deslocamento a esquerda

<<= deslocamento a esquerda por uma constante

>> deslocamento a direita

>>= deslocamento a direita por uma constante

& “E”

| “OU”

~ negação

^ “OU exclusivo”

Operadores booleanos

! não

&& “E”

|| “OU”

Estruturas de repetição: for

A estrutura **for** interage sobre uma lista de objetos, executando um bloco de comandos para cada objeto. Objetos podem ser qualquer coisa que possa ser criada em forma de lista. Por exemplo, para executar determinadas operações a todos os arquivos que possuam extensão “.txt”, o seguinte trecho de código pode ser utilizado:

for

```
for i in *.txt
do
  comando_1
  comando_2
  ...
  comando_n
done
```

for - 2

```
for j in "$var1" "$var2" "$var3" "$var4"
do
  comando_1
  comando_2
  ...
  comando_n
done
```


for 3

```
for k in "1 2 3 4 5 6 7 8 9 10"
do
  comando_1
  comando_2
  ...
  comando_n
done
```

while

O while testa uma condição no início de um laço e fica em loop enquanto essa condição for verdadeira. Ao contrário do laço for, o while é utilizado em situações onde o número de repetições do laço não é conhecido de antemão. Sua sintaxe é:

```
while [condição]
do
  comando_1
  comando_2
  ...
  comando_n
done
```


While - exemplo

```
LIMIT=10
a=1
while [ "$a" -le $LIMIT ]
do
  echo -n "$a "
  let "a+=1"
done
```


until

O until testa por uma condição no início de um laço e fica em loop enquanto esta condição for falsa (seu funcionamento é o oposto do while). Sua sintaxe é:

```
until [condição]
do
  comando_1
  comando_2
  ...
  comando_n
done
```

Until - Exemplo

```
LIMIT=10
var=0
until [ "$var" -ge $LIMIT ]
do
  echo -n "$var "
  let "var+=1"
done
```

Tomada de decisão

Se for preciso verificar o valor de uma variável dentre muitos, podem ser utilizados uma série de encadeamento de testes if e elif. Porém o shell possui a construção case que pode ser utilizada com muito mais facilidade para o casamento de padrões. Sua sintaxe é:

```
case "$variavel" in
  "$condicao1" )
     comando...
     ;;
  "$condicao2" )
     comando...
     ;;
esac
```


Case - Exemplo

```
case "$arch" in
  i386 ) echo "maquina 80386";;
  i486 ) echo "maquina 80486";;
  i586 ) echo "maquina Pentium";;
  i686 ) echo "maquina Pentium2+";;
  *    ) echo "outro tipo de maquina";;
esac
```

Tarefinha!

1. Cálculo de fatorial

   Solicitar a leitura de uma variável (utilizar o comando read) e fornecer como saída o fatorial deste número.

2. Cálculo de IMC (Ìndice de Massa Corporal)

   Sabendo-se que o IMC de uma pessoa consiste em seu peso divido pelo quadrado se sua altura, fazer um script que solicite o peso e a altura de uma pessoa e forneça como saída seu IMC. O script deverá ainda dizer quanto a pessoa deverá perder ou ganhar (e se deverá perder ou ganhar) para ficar com IMC=24,7.

Estudo dirigido

leitura do capítulo 11 do livro
“Advanced Bash-Scripting Guide”.

Aula 04
Comandos internos

Conhecer os comandos internos de um shell

Adquirir conhecimento para o desenvolvimento de scripts mais eficientes

Comandos Internos

Um comando interno (ou builtin) é um comando contido dentro do shell.

Comandos são construídos internamente ao shell por razões de desempenho ou por necessidades específicas de acesso direto à informações internas ao shell.

Comandos de entrada/saída echo

O comando echo imprime na saída padrão uma expressão ou variável. Parâmetros

-e trata caracteres de escape

-n suprime o new line do final

O echo utilizado pelo interpretador de comandos é um comando interno. Não confundir com o programa /bin/echo que tem comportamento similar.

printf

O comando printf é uma melhoria do comando echo, fornecendo uma saída formata. Sua sintaxe é:

printf formato parâmetros

O formato segue o padrão do comando printf da linguagem C.

read

Lê um valor a partir de STDIN (por padrão o teclado) e o armazena em uma variável. Parâmetros:

-s

não mostrar os caracteres digitados

-n N

ler no máximo N caracteres

-p “string”

mostrar “string” antes da leitura

Tarefinha!!

Criar um script que armazene em um arquivo o login e senha digitados, a senha deve está em MD5, usar o md5sum pra isso.

Comandos do sistema de arquivos

cd

Muda o diretório de trabalho (diretório no qual o script irá atuar durante sua execução).

pwd

Retorna o diretório de trabalho corrente.

Comandos que atuam sobre variáveis

let

Executa operações matemáticas sobre variáveis e expressões.

set

Altera o valor de variáveis internas do shell.

Comandos que afetam o comportamento do script

source

O comando source, que tem a mesma funcionalidade do comando ponto (.), importa os comandos do script passado como parâmetro e os executa localmente.

exit

Encerra a execução do script.

exec

Substitui o processo corrente pelo programa passado como parâmetro.

Comandos para controle de jobs

jobs

Lista os processos executando em segundo plano no shell, fornecendo o número do job.

fg

Trás para o primeiro plano um job que está executando em segundo plano.

bg

Coloca em execução em segundo plano um job cuja execução tenha sido suspensa.

kill

Força o encerramento de um processo enviando ao processo um código de encerramento.

killall

Força o encerramento de um processo passado a ele o nome do processo.

Outros comandos

true

Sempre retorna um código de saída de sucesso (valor 0).

false

Sempre retorna um código de saída sem sucesso.

help

Fornece uma tela de ajuda sobre a utilização de comandos internos.

Estudo dirigido

leitura do capítulo 15
do livro “Advanced Bash-Scripting Guide”.

Aula 05
Comandos externos

Conhecer os comandos externos de um shell

Adquirir conhecimento para o desenvolvimento de scripts mais eficientes

Comandos básicos - ls

Comando básico para listar o conteúdo de um diretório. Alguns parâmetros:

-R

lista os subdiretórios recursivamente

-S

ordena pelo tamanho do arquivo

-t

ordena pela hora de modificação

-r

reverte a ordenação

-a

mostra arquivos ocultos (iniciados por .)

-h

mostra em formato mais apropriado para leitura humana

-l

utiliza formato longo para mostrar atributos de arquivos

cat

Envia o conteúdo do arquivo para a saída padrão (por padrão o monitor). O parâmetro -n faz com que as linhas do arquivo sejam numeradas.

tac

Similar ao cat, porém as linhas do arquivo são mostradas da última para a primeira.

rev

Envia o conteúdo do arquivo para a saída padrão (por padrão o monitor), mas mostra as linhas de trás para frente. A ordem das linhas são mantidas, porém são mostrados do último ao primeiro caractere da linha.

cp

Cópia de arquivos. Alguns parâmetros:

-f força a cópia

-i interativo (pergunta antes de sobrescrever)

-R, -r copia diretórios recursivamente

-v mostra os arquivos que estão sendo copiados

mv

Move arquivos. Também utilizado para renomear. Alguns parâmetros:

-f força a movimentação

-i interativo (pergunta antes de sobrescrever)

-v mostra os arquivos que estão sendo movidos

rm

Remove (deleta) arquivos. Alguns parâmetros:

-f força a remoção

-i interativo (pergunta antes de remover)

-R, -r remove diretórios recursivamente

-v mostra os arquivos que estão sendo removidos

rmdir

Remove diretórios vazios.

mkdir

Cria diretório.

chmod

Muda os atributos de arquivos e diretórios.

ln

Cria links para arquivos e diretórios.

man, info

Fornece informações sobre o funcionamento dos comandos externos.

Comandos para manipulação de data e hora

date

Mostra a data e a hora do sistema. Também é utilizados para acertar a data/hora.

time

Mostra estatísticas de utilização de tempo do processo executado como parâmetro.

at

Programa um aplicativo para executar automaticamente em determinada data/hora.

cal

Mostra um calendário.

sleep

Suspende a execução por uma dada quantidade de segundos.

hwclock, clock

Consulta ou acerta a hora no relógio do hardware.

Comandos para processamento de texto

sort

Mostra um arquivo em ordem alfabética.

uniq

Remove linhas duplicadas de um arquivo ordenado.

head

Envia as primeiras linhas de um arquivo para a saída padrão.

tail

Envia as últimas linhas de um arquivo para a saída padrão. Quando utilizado com o parâmetro -f o comando continua mostrando novas linhas que são adicionadas ao arquivo (útil para monitorar arquivos de log).

wc

Conta a quantidade de linhas, palavras e caracteres em um arquivo.

nl

Numera as linhas de um arquivo.

Comandos para arquivamento

tar

Junta vários arquivos em um único arquivo ou dispositivo.

gzip

Faz a compressão de um arquivo. Normalmente utilizado em conjunto com o tar quando é necessário fazer a compressão de mais de um arquivo.

bzip2

Outro programa para compressão de arquivo. Normalmente mais eficiente que o gzip.

zip, unzip

Programas para manipulação de arquivo comprimido no formato zip.

arj, unarj

Programas para manipulação de arquivo comprimido no formato arj.

rar, unrar

Programas para manipulação de arquivo comprimido no formato rar.

Arquivos

file

Identifica o tipo de um arquivo.

which

Mostra em que diretório se encontra determinado arquivo.

diff

Compara 2 arquivos e mostra as diferenças entre eles.

sum, cksum, md5sum, sha1sum, sha256sum

Comandos para gerar hash de arquivos a fim de verificar sua integridade.

more, less

Pagina arquivos e os envia para a saída padrão. O comando less permite um maior controle sobre a paginação.

Comandos do sistema

su

Executa um programa como um usuário substituto ou inicia um shell como outro usuário.

uname

Exibe informações sobre o sistema.

free

Mostra o total de memória RAM e swap utilizada pelo sistema.

du

Mostra o total de espeço em disco utilizado (recursivamente) por arquivos e/ou diretórios.

df

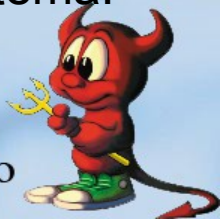
Mostra a utilização dos discos/partições.

dmesg

Mostra todas as mensagens emitidas durante o processo de boot do sistema.

uptime

Mostra por quanto tempo o sistema está em execução.



Estudo dirigido:

leitura do capítulo 16 do
livro “Advanced Bash-Scripting Guide”.

Aula 06

Comandos externos

Substituição de comandos, redirecionamento, pipe e manipulação de strings

Substituição de comandos

A substituição de comandos é utilizada para reatribuir a saída de um comando (ou de múltiplos comandos). A saída do comando pode ser utilizada como argumento para outro comando, ser atribuída a uma variável ou ser utilizada para gerar uma lista de argumentos para um loop for.

A forma tradicional de se utilizar substituição de comandos é executar o comando entre crases (`...`). Por exemplo, o código:

lista=`ls /etc`

coloca na variável lista o conteúdo do ditetório “/etc”.

Utilizando a substituição de comandos também é possível transferir o conteúdo de um arquivo para uma variável:

arq=`cat /etc/passwd`

Redirecionamento

Para todo processo existem sempre 3 arquivos abertos por padrão:

- stdin ? entrada padrão (teclado)
- stdout ? saída padrão (monitor)
- stderr ? saída de erros (monitor)

Estes e quaisquer outros arquivos abertos podem ser redirecionados. Redirecionamento significa capturar a saída de um arquivo, comando, programa, script ou até mesmo um bloco de código em um script e o enviar como entrada para outro arquivo, comando, programa ou script.

Cada arquivo aberto recebe um descritor de arquivo. Os descritores de arquivo para stdin, stdout e stderr são 0, 1 e 2, respectivamente.

O “>” redireciona a saída padrão (stdout) para um arquivo. Se o arquivo não existir ele será criado, caso contrário será sobrescrito e seu conteúdo anterior será perdido.

O “>>” também redireciona a saída padrão (stdout) para um arquivo. Se o arquivo não existir ele será criado. Porém, caso o arquivo exista, a saída do comando irá ser acrescentada ao final do arquivo.

O “2>” redireciona a saída de erros (stderr) para um arquivo. Se o arquivo não existir ele será criado, caso contrário será sobrescrito e seu conteúdo anterior será perdido.

O “2>>” também redireciona a saída padrão (stdout) para um arquivo. Se o arquivo não existir ele será criado. Porém, caso o arquivo exista, a saída do comando irá ser acrescentada ao final do arquivo.

```
ARQ_LOG=script.log
ARQ_ERRO=script.err

echo "Primeira linha enviada ao arquivo de log" > $ARQ_LOG
echo "Segunda linha enviada ao arquivo de log" >> $ARQ_LOG
echo "Terceira linha enviada ao arquivo de log" >> $ARQ_LOG
echo "Quarta linha enviada ao arquivo de log" >> $ARQ_LOG
echo "Quinta linha enviada ao arquivo de log" >> $ARQ_LOG
comando_errado 2> $ARQ_ERRO
```

De uma forma mais geral, para redirecionar do descritor de arquivos M para o descritor de arquivos N, utiliza-se “M>&N”. Por exemplo, “2>&1” redireciona stderr para stdout.

O “<” redireciona um arquivo para a entrada padrão (stdin).

O comando a seguir utiliza o programa sort para ordenar um arquivo. A entrada padrão do sort recebe como redirecionamento o arquivo a ser ordenado (nomes.txt) e a saída padrão é redirecionada para o arquivo que conterá as linhas ordenadas pelo sort (nomes_ordenados.txt).

```
sort <nomes.txt >nomes_ordenados.txt
```

Pipe

O pipe (|) é um mecanismo especial de redirecionamento utilizado para conectar a saída padrão de um processo à entrada padrão de outro processo.

Por exemplo, o código:

```
cat *.txt | sort | uniq > resultado.txt
```

junta todos os arquivos com extensão .txt ordenando suas linhas e retirando as duplicadas.

O processo que se encontra à esquerda do pipe tem sua saída padrão redirecionada automaticamente para a entrada padrão do processo que se encontra à direita do pipe.

Manipulação de strings

Tamanho da string

O tamanho de uma string pode ser obtido por intermédio de ${#string}. Por exemplo:

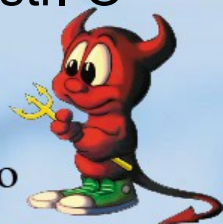
```
echo "A string str possui ${#str} caracteres."
```

Índice

Encontra em uma string a posição numérica do primeiro caractere de uma substring que exista na string. Por exemplo, o comando:

```
str="Talita levou Maria para passear"
substr="Maria"
echo `expr index "$str" "$substr"`
```

irá mostrar o valor 2, pois o primeiro caractere existente tanto em str quanto em substr é o caractere “a”, e encontra-se na segunda posição de str. O primeiro caractere possui índice 1.



Extração de substring

O comando ${string:pos:tamanho} extrai de string uma substring de tamanho caracteres a partir da posição pos. Caso tamanho seja omitido, será extraída a substring iniciando em pos e indo até o final. Por exemplo:

```
str="Talita levou Maria para passear"
substr="Maria"
echo "${str:13:5}"
```

irá mostrar a string “Maria”. O primeiro caractere possui índice 0.

Remoção de substring

${string#substring}
remove a menor porção de string que combinar com o padrão de substring a partir do início da string.

${string##substring}
remove a maior porção de string que combinar com o padrão de substring a partir do início da string.

${string%substring}
remove a menor porção de string que combinar com o padrão de substring a partir do final da string.

${string%%substring}
remove a maior porção de string que combinar com o padrão de substring a partir do final da string.

```
stringZ=abcABC123ABCabc
#       |----|          menor
#       |----------|    maior
echo ${stringZ#a*C}      # 123ABCabc
echo ${stringZ##a*C}     # abc

stringZ=abcABC123ABCabc
#                 ||     menor
#       |-----------|     maior
echo ${stringZ%b*c}      # abcABC123ABCa
echo ${stringZ%%b*c}     # a
```


Troca de substring

${string/substring/substituta}

troca a primeira ocorrência de substring por substituta.

${string//substring/substituta}

troca todas as ocorrências de substring por substituta.

Estudo dirigido:

leitura dos capítulos 5 e 7 do livro “Classic Shell Scripting”.

Aula 07
Exercícios

1) Fazer um script que receba como parâmetro 2 números e retorne:

   1 - caso o primeiro seja maior que o segundo

   0 - caso os dois sejam iguais

   -1 -caso o primeiro seja menor que o segundo

2) Repetir o script anterior para strings.

3) Fazer um script que receba um número como parâmetro e faça a contagem até 0. A contagem deverá ser regressiva se o número for positivo e progressiva se o número for negativo. Deverá ser respeitado o intervalo de 1 segundo entre a apresentação de cada número.

4) Fazer um script que receba como parâmetro 2 strings e retorne em que posição a segunda string aparece dentro da primeira string. O primeiro caractere deverá ser contado como 1 e o valor 0 deverá ser retornado caso a segunda string não apareça dentro da primeira.

5) Fazer um script que receba como parâmetro 3 notas e imprima a média do aluno e se ele está aprovado ou reprovado. Para a média deverão ser consideradas apenas as 2 maiores notas. Para aprovação as duas notas consideradas devem ser maior ou igual a 4 e a média maior ou igual a 6.

6) Fazer um script que receba como parâmetro uma nome e imprima uma série de linhas, onde na primeira linha apareça apenas a primeira letra do nome, na segunda linha as duas primiras letras, e assim por diante. Por exemplo, para o nome “Rita”, imprimir:

R

Ri

Rit

Rita

7) Fazer um script que imprima todos os números múltiplos de 13 entre 1 e 300.

8) Fazer um script que imprima:

Bom dia ? das 06:00 às 11:59

Boa tarde ? das 12:00 às 17:59

Boa noite ? das 18:00 às 5:59

9) Fazer um script que receba 2 números e escreva o dividendo, o divisor, o quociente e o resto.

10) Solicite 2 números e gere uma sequência de 10 números onde cada número (a partir do terceiro) é a soma dos 2 números anteriores.

11) Fazer um script que receba uma data no formato DD/MM/AAAA e forneça separadamente o dia, o mês e o ano.

12) Fazer um script que receba um número e imprima se ele é primo. Caso não seja, mostrar seus divisores.

Metacaracteres

Expressões regulares são constituídas a partir de:

- caracteres normais
- caracteres especiais

Os caracteres especiais (ou metacaracteres) podem ser:

- representantes
- quantificadores
- âncoras

Programação Avançada

Objetivo

Conhecer os fundamentos de expressões regulares

Ser capaz de criar consultas cujos resultados sejam mais rápidos e precisos

Uma expressão regular é um método formal de se especificar um padrão de texto. Uma composição de caracteres com funções especiais que, agrupados entre si e com caracteres literais, formam uma expressão. Essa expressão é interpretada como uma regra, que indicará sucesso se uma entrada de dados qualquer casar com essa regra, ou seja, obedecer exatamente a todas as suas condições. Expressões regulares permitem pesquisar por um texto que se encaixe em determinado critério, permitindo que se escreva uma única expressão que pode selecionar múltiplas strings de dados.

Metacaracteres

Expressões regulares são constituídas a partir de:

- caracteres normais
- caracteres especiais

Os caracteres especiais (ou metacaracteres) podem ser:

- representantes
- quantificadores
- âncoras
- outros

Metacaracteres representantes

Metacaractere	*Mnemônico*	*Função*
.	ponto	um caractere qualquer
[...]	lista	lista de caracteres permitidos
[^...]	lista negada	lista de caracteres proibidos

Metacaracteres quantificadores

Metacaractere	*Mnemônico*	*Função*
?	opcional	zero ou um
*	asterisco	zero, um ou mais
+	mais	um ou mais
{n, m}	chaves	de *n* até *m*

Metacaracteres âncoras

Metacaractere	*Mnemônico*	*Função*
^	circunflexo	início da linha
$	cifrão	final da linha
\b	borda	início ou fim de palavra

Outros metacaracteres

Metacaractere	*Mnemônico*	*Função*
\c	escape	torna literal o caractere *c*
|	ou	ou um ou outro
(...)	grupo	delimita um grupo
\1...\9	retrovisor	texto casado nos grupos 1...9

Não confundir os curingas da linha de comando com metacaracteres de expressão regular. Eles possuem funcionalidades diferentes.

Metacaracteres tipo representante

Os metacaracteres representantes casam com a posição de um único caractere.

O metacaractere . (ponto) casa com qualquer coisa (letra, número, tabulação, @, etc.), inclusive com o caractere ponto.

Exemplos:

Expressão	*Casamento*
n.o	não, nao, n.o, n5o, nAo, ...
e.tendido	estendido, extendido, entendido, ...
12.45	12:45, 12 45, 12345, 12.45, ...
<.>	<B>, <i>, <p>, ...

A lista guarda os caracteres com o qual o casamento é permitido. Caso apareça algum caractere que não conste da lista não haverá o casamento.

Expressão	*Casamento*
n[ãa]o	não, nao
e[sn]tendido	estendido, entendido
12[:.]45	12:45, 12.45, 12 45
<[BIP]>	<B>, <I>, <P>

Dentro da lista todos são caracteres normais. Assim o ponto é considerado como o caractere ponto, não como o metacaractere ponto.

A lista também aceita intervalos. Exemplos:

[0-9] ? [0123456789]
[a-z] ? [abcdefghijklmnopqrstuvwxyz]
[A-Z] ? [ABCDEFGHIJKLMNOPQRSTUVWXYZ]
[3-8] ? [345678]
[d-h] ? [defgh]

Letras maiúsculas, minúsculas e números ? [A-Za-z0-9]

Para que o traço possa ser utilizado em uma lista ele deve estar no final da lista. Assim o padrão [a-g-] casa com as letras de a até g e com o traço.

Os colchetes também devem receber atenção especial. Para representar o colchete que abre não tem problema, ele pode estar em qualquer posição, porém o colchete que fecha deve ser obrigatoriamente o primeiro item da lista, caso exista. A lista [][-] casa com], [, ou -.

Algumas classes especiais

Classe POSIX	*Significado*
[:upper:]	letras maiúsculas
[:lower:]	letras minúsculas
[:alpha:]	letras maiúsculas e minúsculas
[:alnum:]	letras e números
[:digit:]	números
[:xdigit:]	números hexadecimais
[:punct:]	sinais de pontuação
[:blank:]	espaço e tabulação
[:space:]	caracteres brancos (\t\n\r\f\v)
[:cntrl:]	caracteres de controle
[:graph:]	caracteres imprimíveis
[:print:]	caracteres imprimíveis e o espaço

Os colchetes fazem parte da classe, assim [[:upper:]] é uma classe POSIX dentro de uma lista.

As classes POSIX levam em conta a localidade. Assim, no Brasil [:upper;] engloba ABCDEFGHIJKLMNOPQRSTUVWXYZÇÁÀÂÃÉÈÊÕÍÌÎ ÓÔÒÕÚÛÙ...

A lista negada funciona com lógica inversa à lista, ou seja, ela guarda os caracteres com os quais o casamento não é permitido. Por exemplo, [^0-9] significa poder casar com qualquer coisa, exceto dígitos.

Metacaracteres tipo quantificador

Metacaracteres quantificadores indicam o número de repetições permitidas para a entidade imediatamente anterior.

O ? (opcional) indica que a entidade anterior pode ocorrer 0 ou 1 vez.

Expressão	*Casamento*
casas?	casa, casas
fala[r!]?	falar, fala!, fala
</?[BIPbip]>	</B>, </I>, </P>, , , , <B>, <I>, <P>, , ,

O * indica que a entidade anterior pode aparecer quantas vezes for necessário (0, 1 ou mais).

exemplos

Expressão	*Casamento*
3*4	4, 34, 334, 3334, 33334, 333334, ..., 3333333333333333334, ...
bi*p	bp, bip, biip, biiip, ..., biiiiiiiiiiiip, ...
b[ip]*	b, bi, bp, bip, bpipippp, bipiiippi, ...

O .* representa qualquer caractere em qualquer quantidade. O + tem funcionamento parecido com o *, porém ele obriga que case pelo menos uma vez.

Expressão	*Casamento*
3+4	34, 334, 3334, 33334, 333334, ..., 3333333333333333334, ...
bi+p	bip, biip, biiip, ..., biiiiiiiiiiiip, ...
b[ip]+	bi, bp, bip, bpipippp, bipiiippi, ...

O {n,m} significa repetir a entidade anterior um mínimo de n e um máximo de m vezes.

Expressão	*Casamento*
3{1,3}4	34, 334, 3334
bi{2,5}p	biip, biiip, biiiip, biiiiip
no{4}ta	noooota
fu{3,}i	fuuui, fuuuui, fuuuuui, fuuuuuui, ...

Metacaracteres tipo âncora

Metacaracteres do tipo âncora marcam uma posição específica na linha.

O ^ indica que o casamento deve ocorrer no início da linha. Assim ^[0-9] indica que a linha deve ser iniciada com um número. Já ^[^0-9] indica que está sendo procurada uma linha que não se inicia com número.

O $ indica que o casamento deverá ocorrer no final da linha. Assim [0-9]$ indica que a linha deverá terminar com um número.

Expressão	*Casamento*
^$	linha vazia
.....$	5 últimos caracteres de uma linha
^.{15-80}$	linhas com 15 a 80 caracteres

O \b indica uma borda, ou seja, o limite de uma palavra.

Expressão	*Casamento*
dia	dia, diafragma, radial, melodia, bom-dia!
\bdia	dia, diafragma, bom-dia!
dia\b	dia, melodia, bom-dia!
\bdia\b	dia, bom-dia!

Outros metacaracteres

O \ funciona como um caractere de escape. Ele serve para que metacaracteres tenham significado literal. Assim, \. significa o caractere ponto, não o metacaractere ponto.

Para representar um número de CPF com formato nnn.nnn.nnn-nn pode-se utilizar [0-9]{3}\.[0-9]{3}\.[0-9]{3}-[0-9]{2}

O | funciona como um ou, podendo ser escolhida uma opção. Assim boa-tarde|boa-noite procura pela ocorrência de " boa-tarde" ou de "boa-noite". Uma lista também funciona como ou, mas para um caractere somente.

Pode-se juntar vários caracteres ou metacaracteres entre parênteses para formar um grupo.

Expressão	*Casamento*
(ha!)+	ha!, ha!ha!, ha!ha!ha!, ...
(\.[0-9]){3}	.0.6.2, .2.8.9, .6.6.6, ...
boa-(tarde|noite)	boa-tarde, boa-noite
(#|n\.|núm) 6	# 6, n. 6, núm 6
(in|con)?certo	incerto, concerto, certo

O \1...\9 busca um trecho que já tenha casado com um grupo para reutilizá-lo. Assim (quero)-\1 casa com quero-quero. De forma mais ampla, ([A-Za-z]+)-\1 casa com qualquer palavra repetida separada por traço.

Expressão	*Casamento*
(lenta)(mente) é \2 \1	lentamente é mente lenta
((band)eira)nte \1 \2a	bandeirante bandeira banda
in(d)ol(or) é sem \1\2	indolor é sem dor
((((a)b)c)d)-1 = \1,\2,\3,\4	abcd-1 = abcd,abc,ab,a

Função

```
Funcao()
{

}
```

Pode receber $1, $2...

Funcao 2 3

Local permite a criação de variavel local na função.

Avaliação

1. Escrever uma função que receba 2 números e retorne como código de status:

0 se os números forem iguais

1 se o primeiro for menor que o segundo

2 se o primeiro for maior que o segundo

Escrever um script que utilize e teste a função.

2. Escrever uma função que receba como parâmetro um número e imprima a sequência de Fibonacci contendo apenas números menor do que àquele passado como parâmetro. Sequência de Fibonacci = 0 1 1 2 3 5 8 13 21 …, onde o próximo da sequência é a soma dos 2 anteriores.

Grep , egrep, fgrep

grep ? Utiliza expressões regulares básicas para realizar a pesquisa.

egrep ? Utiliza expressões regulares estendidas para realizar a pesquisa.

fgrep ? Versão otimizada do grep para realizar pesquisa por strings. Por não procurar por expressões regulares, seu funcionamento é muito mais rápido que o grep e o egrep.

Sintax

A sintaxe do grep é

grep [opções...] padrão [arquivos...]

O grep lê cada arquivo passado pela linha de comando e quando a linha cada com o padrão pesquisado ele mostra a linha. Quando múltiplos arquivos são passados o nome do arquivo seguido de dois pontos (“:”)precede a linha.

Principais opções

-E ? Pesquisa utilizando expressões regulares estendidas (substitui o egrep).

-F ? Pesquisa utilizando strings fixas (substitui o fgrep).

-i ? Ignora diferenças entre maiúsculas e minúsculas.

-l ? Mostra os arquivos que casaram com o padrão no lugar de mostrar a linha no arquivo.

-q ? Em vez de mostrar os resultados da pesquisa apenas retorna se obteve ou não sucesso na pesquisa.

-s ? Suprime as mensagens de erro.

-v ? Mostra as linhas que não casam com o padrão.

sed

Muitos scripts, notadamente os voltados para gerar mensagens enviadas automaticamente, baseiam-se em um texto padrão que será modificado de acordo com seu destinatário ou finalidade, de forma semelhante a uma “mala direta” de um editor de textos.

O programa mais utilizado para tarefas como esta é o sed (stream editor). As sintaxes mais comuns do sed são:

sed [-n] 'comando_de_edição' [arquivo...]

sed [-n] -e 'comando_de_edição' ... [arquivo...]

sed [-n] -f arquivo_script ... [arquivo...]

O sed recebe um fluxo de texto como entrada e produz o resultado na saída padrão, sem modificar os arquivos originais. O sed é um programa capaz de realizar tarefas complexas, muito além de simples substituição de textos (seu uso mais comum).

As principais opções do sed são:

-e ? Para indicar que serão utilizados múltiplos comandos de edição.

-f ? Ler os comandos a partir de um arquivo.

-n ? Suprimir a impressão normal das linhas modificadas. As linhas deverão ser impressas explicitamente com o comando p.

O uso básico do sed para substituição de textos é por intermédio do comando s, que procura por um texto utilizando expressão regular e o substitui por outro texto. A sintaxe do comando é:

s/expressão_regular/texto_substituto

Para, por exemplo, listar todos os ususários do sistema, pode-se listar o arquivo “etc/passwd” e remover todo o texto que vem após o primeiro dois-pontos (“:”). Isto pode ser feito com:

sed s/:.*// /etc/passwd

O comando s do sed atua somente na primeira ocorrência na linha. Para atuar em todas as ocorrências é preciso utilizar o sufixo g no comando. Por exemplo, para trocar todas as ocorrências da palavra amendoim por castanha em receita.txt, deve-se utilizar:

sed s/amendoim/castanha/g receita.txt

No lugar do sufixo g também pode ser utilizado um número, indicando que deverá ser aquela ocorrência na linha que será substituída. Por exemplo, 2 indica que ira ser substituída apenas a segunda ocorrência da expressão regular em cada linha.

A opção “-n” do sed suprime a impressão das linhas. Assim as linhas a serem impressas devem ser selecionadas explicitamente com o comando p. O comando abaixo mostra somente as linhas que possuam a string “root” no arquivo “/etc/passwd”.

sed -n /root/p /etc/passwd

cat texto.txt | sed p

- Dobrar as linhas

sed 2p frutas1.txt

- Dobra apenas a segunda linha

sed -n 2p frutas1.txt

- Exibe apenas a segunda linha

Um monte de opção

= imprime o número da linha atual do [ARQUIVO]

inicia um comentário

! inverte a lógica do comando

; separador de comandos

, separador de faixas de endereço

{ início de bloco de comandos

} fim de bloco de comandos

s substitui um trecho de texto por outro

y traduz um caractere por outro

i insere um texto antes da linha atual

c troca a linha atual por um texto

a anexa um texto após a linha atual

g restaura o [TEXTO] contido no ESPAÇO RESERVA (sobrescrevendo)

G restaura o [TEXTO] contido no ESPAÇO RESERVA (anexando)

h guarda o [PADRÃO] no ESPAÇO RESERVA (sobrescrevendo)

H guarda o [PADRÃO] no ESPAÇO RESERVA (anexando)

x troca os conteúdos dos ESPAÇO PADRÃO e RESERVA

: define uma marcação

b pula até uma marcação

t pula até uma marcação, se o último s/// funcionou (condicional)

d apaga o [PADRÃO]

D apaga a primeira linha do [PADRÃO]

n vai para a próxima linha

N anexa a próxima linha no [PADRÃO]

q finaliza o Sed imediatamente

É possível restringir as linhas que o sed irá atuar prefixando o comando com um endereço. As formas de se endereçar linhas são:

expressões regulares ? Colocando uma expressão regular antes de um comando ele irá atuar somente sobre as linhas que casarem com o padrão.

última linha ? Colocando o símbolo $ antes do comando estará limitando a execução do comando à ultima linha fornecida. O $ deve ser protegido por uma contrabarra (“\”) ou por aspas simples.

números de linha ? Coloca-se o número da linha antes do comando.

faixas ? Pode-se especificar uma faixa de linhas separando os endereços por vírgula.

expressões regulares negadas ? Acrescentando um caractere ! após uma expressão regular serão selecionadas as linhas que não casarem com a expressão regular. O ! deve ser protegido por uma contrabarra (“\”) ou por aspas simples.

Segue abaixo exemplo de um script que recebe um valor n e um nome de arquivo como parâmetros e mostra as n primeiras linhas do arquivo.

```
contagem=$1
sed -n 1,${contagem}p "$2"
```