PROGRAMAÇÃO EM SHELL SCRIPT

UNIDADE 4 - PROGRAMAÇÃO EM SHELL SCRIPT

Andrea Karina Garcia

Introdução

Olá, caro estudante! Seja bem-vindo a esta unidade de **Programação em Shell Script**.
Você é um entusiasta de Linux? Desenvolver, criar e, até mesmo, brincar com as linhas de comando é uma interação característica e peculiar desse sistema operacional tão simpático. Mesmo que você não tenha tanta proximidade dele, ainda assim pode executar algumas tarefas com facilidade por meio de sua interface convidativa, apenas pontando e clicando. Quanto mais você usa e aprende sobre a linha de comando, mais você cria maiores e novas possibilidades, sendo que a própria linha de comando é um programa, o Shell. Muitas distribuições Linux usam o Bash, e é nesse universo que você, de fato, está adentrando, ao inserir um comando.
O Bash Script é bastante utilizado para o suporte à administração de redes, principalmente em ambiente Unix, para automatizar a administração de um sistema (COSTA, 2010). As linguagens utilizadas para escrever scripts de Shell são chamadas de "Shell Scripting".
Os scripts permitem programar comandos em cadeia, a partir de arquivos de texto contendo comandos do Unix e comandos especiais do Shell para variáveis, testes, laços, funções e comentários. Mas, na prática, como isso funciona? Você pode, por exemplo, chamar um comando, como o DATE +%F, e usá-lo como parte de um esquema de nomeação de arquivos (para saber mais opções do comando DATE, digite, em seu terminal Linux, "**date --help** "). Os scripts também permitem o uso de funções de programação, como *loops* e instruções. Esse é o verdadeiro fascínio de um script!
Nesta unidade, você estudará o editor Vi, as funções para o encadeamento de rotina Shell, os comandos e variáveis internas, entre outros detalhes pertinentes ao assunto.
Vamos lá? Bons estudos!

4.1 Declarando funções para o encadeamento de rotinas em Shell

Saiba que o Shell Bash é usado como padrão na maior parte das distribuições Linux. Ele está disponível para os sistemas operacionais Mac OS e Cygwin. O sistema Cygwin consiste em um emulador Linux que é executado sobre o sistema operacional Windows. Seus scripts funcionarão, de maneira geral, com pequenos ajustes (ou sem necessidade de adaptações) em ambas as plataformas. Por outro lado, sugerimos testar os scripts antes de colocá-los em funcionamento. Os scripts são executados como programas, mas, para isso, eles necessitam de permissões apropriadas. Saiba mais a seguir!

4.1.1 O que é uma função Shell?

O Shell tem seus próprios comandos, variáveis e funções. O papel de uma função é segmentar vários códigos em um script. Tenha em mente que usar uma função é como colocar vários scripts em um só script.

> Funções são estruturas que reúnem comandos na forma de blocos lógicos, permitindo a separação do programa em partes. Quando o nome de uma função é chamado dentro do script como um comando, todos os comandos associados a esta função são executados (PET-Tele UFF, 2004, p. 59).

VOCÊ SABIA?

Você sabia que um programa pode ser escrito em um editor de sua preferência, como Vi, kWrite, KEdit, entre outros? Em sua essência, os scripts são arquivos de texto simples, que serão salvos como texto comum, independentemente de sua escolha.

Como definir uma função dentro do Shell Script? Bem, a sintaxe utilizada é muito semelhante à sintaxe da linguagem de Programação C. Assim, temos inicialmente que definir um nome para a função; nome este que será utilizado em outras partes do script para a evocação da função implementada. Uma função Shell não admite passagem de parâmetros em sua definição de interface, e, desse modo, após o nome da função, sempre virá o par formado pela abertura e fechamento de parênteses. Similar à linguagem C, o corpo da função é delimitado pelas chaves "{" (denota o início do bloco) e "}" (caractere relativo ao final do bloco). Para um melhor entendimento, vamos mostrar, a seguir, o formato básico de uma função Shell script:

```
NomeDaFuncao()
{
corpo_da_função → neste espaço virá o código da função
}
```

Convém salientar que, em algumas ocasiões, em virtude do tipo do Shell a ser utilizado, surge a necessidade de colocar a palavra "**function**" precedendo o nome da função, ficando: "**function NomeDaFuncao()**".

Tenha em mente que a principal vantagem de usar funções é a possibilidade de organizar o código do programa em módulos. Com a finalidade de se aumentar a modularidade, é possível que um script seja evocado a partir de um outro script, como se fosse uma função externa. Para que isso seja feito, deve-se seguir a seguinte sintaxe: ". nome_arquivo_script" (PET-Tele UFF, 2004). Por exemplo, imagine que um script codificado no arquivo ScriptTeste.sh tenha que ser evocado a partir da função "Teste()". A implementação desse cenário fica assim codificada:

```
Teste()
{
. ScriptTeste.sh
}
```

Nesse caso, o caractere "**.**" indica que o arquivo "**ScriptTeste.sh**" se localiza na mesma pasta em relação ao arquivo que contém a função "**Teste()**". Porém, caso "**ScriptTeste.sh**" seja localizado em uma pasta distinta, o caractere "**.**" deve ser substituído pelo caminho referente à pasta.

O Linux permite, ainda, a execução em modo de depuração. Para tanto, a execução do script deve ser realizada utilizando-se o parâmetro "-x". Tal parâmetro faz com que a execução do script ocorra passo a passo (PET-Tele UFF, 2004). O parâmetro "-x" é passado ao comando SH da seguinte forma: "**$sh -x script.sh**".

VOCÊ QUER VER?

Para a execução de um script em Shell, em algumas ocasiões, temos que realizar tarefas como mudar as permissões do arquivo que contém o código Shell Script. Para ter uma visão inicial de como proceder, você poderá acessar o vídeo disponível em: <https://www.youtube.com /watch?v=zZXxpXnxLj4.

Do ponto de vista do bash, saiba que uma função Shell é um script dentro de outro script. Assim, as funções são capazes de implementar as mesmas rotinas que um script, com a vantagem de que elas poderão ser reutilizadas em outros scripts.

4.2 Blocos de comandos e variáveis internas

As variáveis são a essência da programação, pois permitem que um programador armazene dados, alterando-os e reutilizando-os em scripts. Sigamos juntos, nessa caminhada, para aprendermos um pouco mais sobre o Shell.

4.2.1 O que são variáveis Shell?

Bem, variáveis são estruturas que armazenam dados, como uma espécie de atalho. Cada variável tem um nome, um tipo e um tamanho, com significado dependente da linguagem de programação utilizada (COSTA, 2010). O bash reconhece uma variável quando ela começa com $, sendo ela, portanto, utilizada da seguinte forma (PET-Tele UFF, 2004):

$nomevar=valor

Nesse exemplo, uma variável de nome "nomevar" foi instanciada com "valor". No caso da primeira ocorrência, para a criação da variável não se utiliza o símbolo "$" precedendo a variável, uma vez que o símbolo "$" é utilizado em variáveis previamente criadas. Dessa forma, percebemos que, para referenciar uma variável, é preciso utilizar o símbolo "$". Outro detalhe que deve ser mencionado consiste no fato de que é aconselhável não deixar espaços em branco entre o nome da variável, o símbolo de atribuição ("=") e o valor a ser atribuído à variável.

Será que existe diferenciação entre variáveis locais e globais, como acontece, por exemplo, na linguagem C? Sim, na programação em Shell Script, também poderemos fazer essa diferenciação. No exemplo anterior, a variável "**nomevar**" foi definida como uma variável local. A criação de variáveis globais é realizada por meio do identificador "**export**", da seguinte forma (PET-Tele UFF, 2004):

export nomevar

ouexport nomevar=valor

Para uma melhor abstração em relação à criação e à utilização de variáveis dentro do escopo de Shell Script, vamos ver o exemplo a seguir:

```
#!/bin/bash
NomeScript=$0
echo "O nome do script eh $NomeScript"
```

Perceba que temos a criação de uma variável denominada "**NomeScript**" recebendo como valor inicial "**$0**". A variável "**$0**" representa o primeiro argumento passado como parâmetro no comando de linha, denotando, nesse caso, o próprio nome do arquivo de script. Os demais parâmetros passados como argumentos ao script poderão

ser referenciados como "**$1**", "**$2**" e assim sucessivamente. Ainda em relação ao exemplo, temos, na última linha, a impressão na tela utilizando-se o comando ECHO.
Para a definição dos nomes das variáveis, valem algumas regras também pertencentes às linguagens de programação, dentre as quais podemos destacar (PET-Tele UFF, 2004) as que você poderá ver, clicando nos ícones a seguir.

O nome de todas as variáveis deve ser iniciado por letra ou underline.
Somente são permitidos caracteres alfanuméricos.
Espaços em branco, acentos ou caracteres especiais não são permitidos.

Uma variável é apagada quando o comando UNSET for acionado (PET-Tele UFF, 2004):
$unset nomevar
Utilizando o comando SET, é possível visualizar as variáveis locais, ao passo que com o comando ENV, as variáveis globais podem ser vistas.

4.2.2 Variáveis do sistema

Existem algumas variáveis que são próprias do sistema e outras que são inicializadas diretamente pelo Shell. Confira, a seguir, alguns exemplos (PET-Tele UFF, 2004, p. 63-64):

- **HOME –** Contém o diretório home do usuário.
- **LOGNAME –** Contém o nome do usuário atual.
- **IFS –** Contém o separador de campos ou argumento (Internal Field Separator). Geralmente, o IFS é um espaço, tab, ou nova linha, mas é possível mudar para outro tipo de separador.
- **PATH –** Armazena uma lista de diretórios onde o Shell procurará pelo comando digitado.
- **PWD –** Contém o diretório corrente.
- **PS1 –** Esta é denominada "Primary Prompting String". Ou seja, ela é a string que está no prompt que vemos no Shell. Geralmente, a string utilizada é: \u@\h:\w $.
- **PS2 –** Esta é denominada "Secondary Prompting String". Ela armazena a string do prompt secundário. O padrão usado é o caractere >. Mas pode ser mudado para os caracteres mostrados acima.
- **MAIL –** É o nome do arquivo onde ficam guardados os e-mails.
- **COLUMNS –** Contém o número de colunas da tela do terminal.
- **LINES –** Contém o número de linhas da tela do terminal. Existem muitas outras variáveis que são descritas na página do manual do Bash, na seção "Shell Variables".

Confira, agora, o significado desses caracteres e de outros principais.

Caractere	Descrição
\s	O nome do Shell.
\u	Nome do usuário que está usando o Shell.
\h	O *hostname*.
\w	Contém o nome do diretório corrente desde a raiz.
\d	Mostra a data no formato: dia_da_semana mês dia.
\nnn	Mostra o caractere correspondente em números na base octal.
\t	Mostra a hora atual no formato de 24 horas, HH:MM:SS.
\T	Mostra a hora atual no formato de 12 horas, HH:MM:SS.
\W	Contém somente o nome do diretório corrente.

Figura 1 - Principais caracteres Shell.
Fonte: PET-Tele UFF, 2004, p. 63-64.

Existem várias maneiras de se instanciar uma variável. Além de ser associada a uma frase, a um número ou a alguma outra variável, podemos, por exemplo, misturar valores com variáveis. Nesse caso, devemos fazer uso das aspas, como no exemplo a seguir:

var2 = "Atribuindo um novo valor. Anexando o valor $var1"

Por outro lado, se em vez das aspas utilizássemos apóstrofos, o conteúdo da variável "**$var1**" não seria anexado, e apareceria literalmente a palavra "*$var1*" na tela. Isso seria a consequência da utilização dos apóstrofos, que faz com que o conteúdo a ser atribuído não seja interpretado, ou seja, o valor entre apóstrofos é atribuído à variável sem qualquer tipo de manipulação. Por fim, o conteúdo pode aparecer entre crases, e, nesse caso, ocorre a interpretação de um comando e o seu respectivo retorno de valor.

VAMOS PRATICAR?

Realize uma pesquisa na Internet e selecione a melhor definição de "funções", referenciando sua fonte.

4.3 Editores de texto e as expressões regulares

Para se editar um Shell Script, pode ser utilizado qualquer editor que permita a gravação de arquivos textuais. Assim, podemos citar os editores Vi, Vim, XEmacs, kWrite e KEdit. Deve-se gerar arquivos textuais porque o Shell, por exemplo, do Linux, deve ler o arquivo para realizar a interpretação dos comandos nele contidos. Nesse caso, o Shell não teria condições de manipular um arquivo binário formatado, como é o caso, por exemplo, dos arquivos "*.odt*" ou "*.doc*" gerados pelo OpenOffice Writer.

4.3.1 Editor de texto Vi

Como você já deve saber, o Vi é um editor de textos, e seu nome é uma abreviação de "*visual interface*". Esse editor, desenvolvido na Universidade da Califórnia, tem sua história dentro do universo Unix, a qual deve ser considerada para que ele seja compreendido (COELHO, 2002). O *software* foi desenvolvido em um ambiente que os terminais dos usuários eram "teletipos" (ou algum outro terminal lento do tipo "*hardcopy*") e os monitores de vídeo, em geral, não eram utilizados. Para esse tipo de ambiente, um editor adequado seria um orientado por linhas, em que os usuários pudessem visualizar e trabalhar com uma linha de cada vez.

Segundo Coelho (2002, p. 5), "o Vi é um editor de textos, não muito amigável à primeira vista, mas possui vantagens que superam a dificuldade inicial de adaptação a seus comandos". Ainda de acordo com o autor, os desenvolvedores tinham a intenção de fazer um editor que não exigisse do usuário que movimentasse muito as mãos para operá-lo. O Vi tem um número extenso de comandos, o que dificulta sua utilização, sendo um "programa complexo e completo" (COELHO, 2002, p. 6). No entanto, tenha em mente que não é necessário dominar muitos desses comandos para começar a trabalhar com o editor. Veremos, a seguir, alguns de seus principais comandos.

Para iniciar, digite "vi" na linha de comandos do sistema operacional. Para um arquivo nomeado após o comando, se o arquivo não existir, o editor abrirá um novo documento. Se o arquivo existir, o Vi o abrirá e permitirá a sua edição (COELHO, 2002).

O Vi pode ser iniciado a partir dos seguintes comandos:

COMANDO	Descrição
vi *arq*	Inicia o Vi e abre o arquivo *arq*.
vi *arq01 arq02*	Inicia o Vi e abre os arquivos *arq01* e *arq02*.
view *arq*	Inicia o Vi e abre o arquivo *arq* em modo somente de leitura (*read only*).
vi –R *arq*	Inicia o Vi e abre o arquivo *arq* em modo somente de leitura (*read only*).
vi –t *tag*	Procura por uma *tag* e inicia a edição a partir de sua definição.
vi –w *n*	Abre o Vi com uma janela com "n" linhas.
vi + *arq*	Abre o arquivo *arq* e posiciona o cursor na última linha.
vi +*n arq*	Abre o arquivo *arq* e posiciona o cursor na linha "n".
vi –c *comando arq*	Abre o arquivo *arq* e executa o *comando* (geralmente um comando de busca).
vi +/*padrão arq*	Abre o arquivo *arq* fazendo automaticamente uma busca de *padrão*.

Figura 2 - Comandos do Vi no Shell.
Fonte: COELHO, 2002, p. 11

O Vi pode operar tecnicamente de três formas. Contudo, neste momento, vamos tratar apenas de duas. Clique nos ícones a seguir.

Modo de comando (padrão).
Modo de inserção.

O Vi, por padrão, inicia em modo de comando. Nele, as teclas expressam um comando associado. Para alterar o modo de operação, há dois comandos mais utilizados.

COMANDO	Descrição
a	Entra em modo de inserção de caracteres, posicionando o cursor à direita do caractere em que ele está sobre.
i	Inicia a inserção de caracteres a partir da esquerda do caractere em que o cursor está sobre.

Figura 3 - Comandos do Shell para alterar o modo de operação.
Fonte: COELHO, 2002, p. 12.

No modo de inserção, os caracteres do teclado que são digitados aparecem no documento. Para corrigir, pode-se utilizar a tecla "backspace" e "delete". O "tab", da mesma forma, é usado para tabular. Para escrever um texto, ao quebrar a linha, basta teclar "enter". Para voltar ao modo de comandos, deve-se teclar "esc" (COELHO, 2002).

COMANDO	Descrição
esc	Retorna ao modo de comando.

Figura 4 - Comandos do Shell.
Fonte: COELHO, 2002, p. 12.

Se seu teclado não possui a tecla "esc", Coelho (2002) indica que você faça uso da seguinte estratégia:

- digite "^[";
- se o seu teclado tiver a tecla <meta>, tente pressioná-la;
- se o seu teclado não tiver a tecla <meta>, tente <alt>, <diamond>, <start> ou <menu> como uma tecla <meta>;
- se nada funcionar, tente ^3'
- alguns terminais DEC também utilizam a tecla F11 como <esc>.

Como em qualquer edição, é recomendável que gravações periódicas sejam realizadas. Para tanto, pode-se utilizar o comando :W (w = *write*, ou seja, escrever), que fará com que o arquivo seja salvo sem fechar o Vi. Porém, caso haja a necessidade de fechar o editor, pode-se utilizar os comandos :WQ ou :ZZ, conforme detalha a tabela a seguir (COELHO, 2002):

COMANDO	Descrição
:w	Grava o texto e continua editando. Caso se informe um nome após o comando, o texto será gravado com esse nome.
ZZ	Grava o texto, se alterado, e sai do Vi.
:wq	Grava o texto e sai do Vi.
:w!	O arquivo será gravado mesmo se for marcado somente para leitura.
:q	Sai do Vi. Se o texto foi alterado e não foi salvo, você será alertado.
:q!	Força a saída do Vi sem gravar o texto, mesmo que este tenha sofrido alterações desde a última gravação.

Figura 5 - Comandos do Shell para gravar um arquivo.
Fonte: COELHO, 2002, p. 14.

Para se deslocar pelo texto, você pode digitar um só comando para fazer a mesma ação (COELHO, 2002).

COMANDO	Descrição
h	Desloca o cursor para a esquerda.
j	Desloca o cursor para baixo.
k	Desloca o cursor para cima.
l	Desloca o cursor para a direita. Embora não seja muito comum, antes dos comandos h, j, k, l, pode-se informar o número de vezes que eles serão executados.
[*n*]^b	Volta uma página no texto. Você pode inserir a quantidade de páginas a voltar.
[*n*]^f	Avança uma página no texto. Da mesma forma que o comando ^b, você pode informar quantas páginas deseja avançar.

Figura 6 - Comandos do Shell para deslocamento.
Fonte: COELHO, 2002, p. 15.

No caso de o ambiente no qual você estiver editando o texto não contar com as setas de deslocamento (para direita, para baixo, para cima e para a esquerda), estas poderão ser substituídas pelas teclas correspondentes às letras “h”, “j”, “k” e “l”, respectivamente (COELHO, 2002).

VOCÊ QUER LER?

Como estudante de um curso de tecnologia e futuro profissional da área, é importante problematizar suas aplicações no contexto social, você não concorda? “Do androids dream of eletric chip?” é um romance de ficção científica escrito por Philip K. Dick, em 1968. A obra foi adaptada para o cinema por Ridley Scott, com o título de “Blade Runner”, em 1982. Sugerimos tanto o livro quanto a sua adaptação.

4.3.2 Expressões Regulares

Uma Expressão Regular (ER), segundo Jargas (2004, p. 2), “é um método formal de se especificar um padrão de texto”. São padrões utilizados para selecionar combinações de caracteres com funções especiais, chamados “metacaracteres”. Agrupados entre si, eles formam uma expressão. Essa expressão é testada em textos e retorna sucesso caso esse texto obedeça exatamente a todas as condições. Diz-se, nesse caso, “que o texto ‘casou’ com a expressão” (JARGAS, 2004, p. 2).

As Expressões Regulares servem para especificar um padrão específico existente entre trechos de código num arquivo, e para efetuar alterações. Os métodos de pesquisa acabam sendo aprimorados, facilitando o trabalho do administrador ou usuário. Cada caractere especial tem uma função que refina ainda mais a busca, tornando a fórmula mais lógica. Num exemplo rápido, [rgp]ato pode casar “rato”, “gato” e “pato” (JARGAS, 2004).

As ERs são úteis para buscar ou validar textos variáveis, como:

- data;
- horário;

- número IP;
- endereço de e-mail;
- endereço de Internet;
- declaração de uma função();
- dados na coluna N de um texto;
- dados que estão entre <tags></tags>;
- número de telefone, RG, CPF, cartão de crédito (JARGAS, 2004).

Vários editores de texto e linguagens de programação têm suporte para ERs. Para estudar de maneira mais detalhada as ERs, observe o exemplo de Jargas (2004) a seguir, o qual servirá de base para as próximas exposições:

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
ntp:x:38:38::/etc/ntp:/sbin/nologin
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
carlos:x:500:500:carlos:/home/carlos:/bin/bash
ana:x:501:501:Ana Paula Moreira:/home/ana:/bin/bash
acs:x:502:502:Antonio Carlos Silva:/home/acs:/bin/bash
$
```

Figura 7 - Arquivo de senhas dos usuários em um sistema operacional Linux. Esse arquivo geralmente tem o seguinte caminho para o seu acesso: "/etc/passwd".
Fonte: JARGAS, 2004, p. 4.

O exemplo, perceba, mostra várias linhas, uma para cada usuário. O caractere responsável por separar as informações é o ":". A letra "x" indica que todas as senhas estão protegidas em outro arquivo, sendo que a primeira linha é referente ao administrador do sistema, o *root*. Em vez de listar o conteúdo do arquivo todo, a fim de saber os dados do usuário *root*, você pode usar o comando GREP para a pesquisa. Dessa forma, basta informar a palavra-chave e o nome do arquivo da busca (JARGAS, 2004).

```
$ grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
$
```

Além da linha *root*, o GREP retornou a linha do usuário "operator". Como o diretório do usuário operator também é *home*, os dois resultados são válidos. Para sermos mais específicos, você pode fazer uso das expressões regulares. Um metacaractere, portanto, é uma ferramenta que pode auxiliar você a especificar sua pesquisa, informando padrões e posições impossíveis de se especificar usando apenas caracteres corriqueiros. Observe o quadro com a relação dos metacaracteres:

Meta	Nome	Posicionamento
^	circunflexo	Representa o começo da linha
$	cifrão	Representa o fim da linha
		Texto
[abc]	lista	Casa as letras 'a' ou 'b' ou 'c'
[a-d]	lista	Casa as letras 'a' ou 'b' ou 'c' ou 'd'
[^abc]	lista negada	Casa qualquer caractere, exceto 'a', 'b' e 'c'
(esse|aquele)	ou	Casa as palavras 'esse' ou 'aquele'
		Quantidade I
a{2}	chaves	Casa a letra 'a' duas vezes
a{2,4}	chaves	Casa a letra 'a' de duas a quatro vezes
a{2,}	chaves	Casa a letra 'a' no mínimo duas vezes
		Quantidade II
a?	opcional	Casa a letra 'a' zero ou uma vezes
a*	asterisco	Casa a letra 'a' zero ou mais vezes
a+	mais	Casa a letra 'a' uma ou mais vezes
		Curingas
.	ponto	Casa um caractere qualquer
.*	curinga	Casa qualquer coisa, é o tudo e o nada

Figura 8 - A relação dos metacaracteres.
Fonte: JARGAS, 2004, p. 13.

Veja que o primeiro metacaractere da lista é o circunflexo "^", o qual simboliza o início da linha. Ele pode ser usado para resolver o problema anteriormente exposto, e obter somente a linha do usuário *root* (JARGAS, 2004).

```
$ grep ^root /etc/passwd
root:x:0:0:root:/root:/bin/bash
$
```

Do comando GREP foi passado a uma ER, composta pelo metacaractere "^" seguido da palavra "root". O cifrão "$" é posicionado ao fim de uma linha e representa seu fim. De acordo com o exemplo disposto aqui, o último campo de cada linha do arquivo "passwd", é o Shell de *login* do usuário. Outra função do "$" é pesquisar todos os usuários que usam o bash como Shell, além do *root* (JARGAS, 2004).

```
$ grep bash$ /etc/passwd
root:x:0:0:root:/root:/bin/bash
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
carlos:x:500:500:carlos:/home/carlos:/bin/bash
ana:x:501:501:Ana Paula Moreira:/home/ana:/bin/bash
acs:x:502:502:Antonio Carlos Silva:/home/acs:/bin/bash
$
```

A ER Bash$ procura pela palavra "bash" no final da linha, ou pela palavra "bash" seguida de um fim de linha. Tenha em mente que a junção do circunflexo com o cifrão (^$) serve para encontrar linhas em branco. Já os colchetes representam o metacaractere lista **[ ]**. Para fazer uso desse recurso, você precisa colocar entre colchetes todos os caracteres que podem aparecer em uma determinada posição. Podem ser tantos quanto seja necessário. Observe o exemplo a seguir (JARGAS, 2004):

```
$ grep '[Cc]arlos' /etc/passwd
carlos:x:500:500:carlos:/home/carlos:/bin/bash
acs:x:502:502:Antonio Carlos Silva:/home/acs:/bin/bash
$
```

Toda a lista, que compreende os colchetes e seu conteúdo, vale apenas para uma posição, um caractere. A ER "[Cc]arlos" serve para pesquisar por "Carlos" e "carlos", ao mesmo tempo. Portanto, "a lista, por maior que seja, representa apenas um caractere" (JARGAS, 2004).

O ponto é o metacaractere que representa qualquer letra. E mais: na verdade, ele significa qualquer caractere, pois além de letras ele pode representar um número, um símbolo ou um "tab". Às vezes, é necessário representar um caractere numa certa posição, como para procurar usuários onde a segunda letra do *login* seja uma vogal, como "mario", e "jose". Não importa qual é a primeira letra, pode ser qualquer uma, mas a segunda deve ser uma vogal (JARGAS, 2004).

Para representar a quantidade de repetições do caractere ou metacaractere anterior, fazemos uso das chaves. A utilização das chaves na expressão regular pode ser observada no exemplo a seguir. Tal exemplo tem a função de realizar uma pesquisa por qualquer repetição de um número por pelo menos três vezes e, também, com um tamanho mínimo de três dígitos (JARGAS, 2004).

```
$ egrep '[0123456789]{3,}' /etc/passwd
games:x:12:100:games:/usr/games:/sbin/nologin
carlos:x:500:500:carlos:/home/carlos:/bin/bash
ana:x:501:501:Ana Paula Moreira:/home/ana:/bin/bash
acs:x:502:502:Antonio Carlos Silva:/home/acs:/bin/bash
$
```

Quando procuramos por dois trechos específicos na mesma linha, utilizamos o curinga: .*. Um exemplo é procurar os usuários que começam com vogal e usam o Shell Bash (JARGAS, 2004).

```
$ egrep '^[aeiou].*bash$' /etc/passwd
ana:x:501:501:Ana Paula Moreira:/home/ana:/bin/bash
acs:x:502:502:Antonio Carlos Silva:/home/acs:/bin/bash
$
```

No exemplo a seguir, temos uma Expressão Regular cujo modo de operação é análogo a um operador OR lógico, ou seja, o seu retorno é “verdade” se pelo menos uma das expressões de entrada tiver o valor lógico “verdade”. Outros operadores lógicos, tais como o AND, também poderiam ser utilizados nas expressões regulares (JARGAS, 2004).

```
$ egrep '^(ana|carlos|acs):' /etc/passwd
carlos:x:500:500:carlos:/home/carlos:/bin/bash
ana:x:501:501:Ana Paula Moreira:/home/ana:/bin/bash
acs:x:502:502:Antonio Carlos Silva:/home/acs:/bin/bash
$
```

Na Expressão Regular anterior, perceba, há a combinação das linhas dos três usuários especificados. No caso, o caractere "^" denota o início de uma linha. Assim, as palavras “ana”, “carlos” ou “acs” devem aparecer no início das linhas buscadas. Outros metacaracteres também poderão usados, tais como “*”, “+” e “?”. A tabela a seguir, extraída de Jargas (2004), explicita suas funcionalidades.

Meta	Nome	Equivalente	Descrição
?	Opcional	{0,1}	Pode aparecer ou não.
*	Asterisco	{0,}	Pode aparecer em qualquer quantidade.
+	Mais	{1,}	Deve aparecer no mínimo uma vez.

Figura 9 - Descrição das funcionalidades dos metacaracteres "?", "*" e "+" no contexto das Expressões Regulares.
Fonte: JARGAS, 2004, p.10.

Caso o caractere "^" apareça logo após o caractere "[", temos a negação dos caracteres envolvidos em uma lista. Por exemplo, caso a expressão regular seja:

$ grep '^[^aeiou]' /etc/passwd

Significa que pode ser qualquer caractere, com exceção das vogais. Convém salientar que, quando referenciamos "qualquer caractere com exceção das vogais", estamos permitindo que caracteres de controle de formatação, tais como o caractere de tabulação e o espaço em branco, sejam referenciados.

VOCÊ O CONHECE?

George Boole (1815-1864) foi um matemático e filósofo britânico, criador da Álgebra Booleana, alicerce da computação moderna. A importância de sua lógica simbólica só foi reconhecida anos depois. Os trabalhos de Alfred North Whitehead e Bertrand Russel, na década de 1910, que convenceram a comunidade científica de sua importância.

Outra possibilidade que a lista tem é a indicação de intervalos. Para tanto, você deve colocar um hífen entre duas letras, assim, ele será expandido para todas as letras existentes no intervalo. Observe a tabela ASCII a seguir:

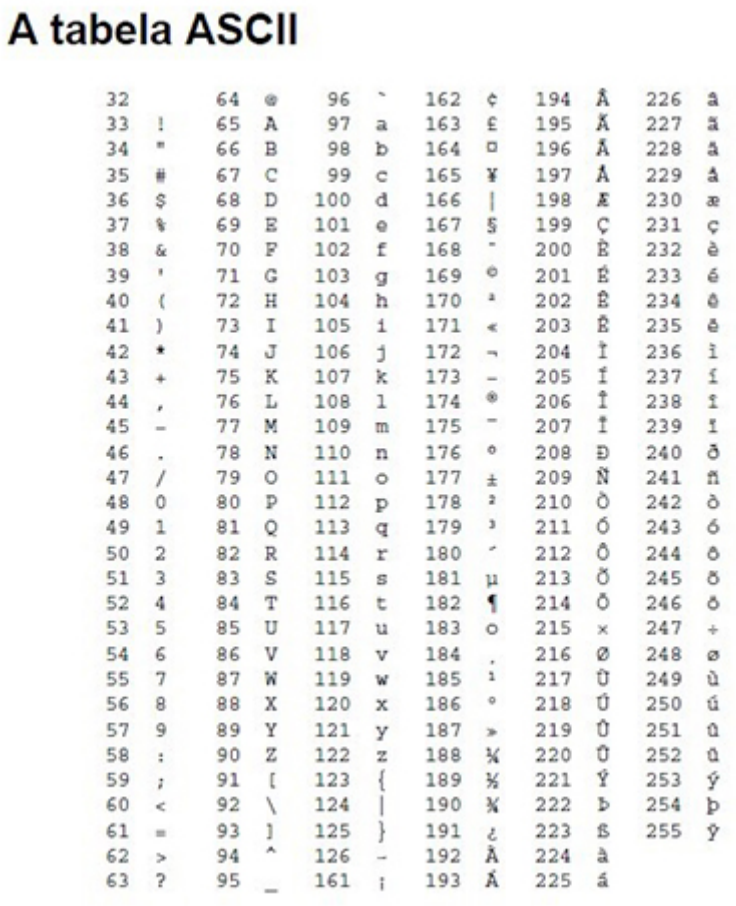

Figura 10 - A tabela ASCII.
Fonte: JARGAS, 2004, p. 12.

Apesar de os metacaracteres utilizados nas Expressões Regulares também aparecerem na manipulação de comandos do Shell dos sistemas operacionais ou, ainda, nas implementações dos scripts Shell, eles podem ter denotações distintas. Expressões Regulares são manipuladas por intermédio de programas como GREP, SED e AWK. Por sua vez, os curingas podem ser usados, por exemplo, na linha de comando dos sistemas operacionais, para possibilitar a expansão dos nomes de arquivos (JARGAS, 2004).

Shell	Expressões Regulares (ERs)
*	.*
?	.
{,}	(|)

Figura 11 - Equivalências e diferenças entre os metacaracteres utilizados nas Expressões Regulares e os utilizados nas linhas de comando de um sistema operacional como o Linux.
Fonte: JARGAS, 2004, p. 13.

A sintaxe das Expressões Regulares pode ser diferente da do EGREP, dependendo do aplicativo utilizado. No SED e AWK, por exemplo, as chaves e o “ou” devem ser escapados para serem especiais. Observe a tabela a seguir (JARGAS, 2004):

Programa	opc	mais	chaves	borda	ou	grupo
awk	?	+	-	-	|	()
egrep	?	+	{,}	\b	|	()
emacs	?	+	-	\b	|	\(\)
find	?	+	-	\b	|	\(\)
gawk	?	+	{,}	\<\>	|	()
grep	\?	\+	\{,\}	\b	|	\(\)
sed	\?	\+	\{,\}	\<\>	|	\(\)
vim	\=	\+	\{,}	\<\>	|	\(\)

Figura 12 - Diferentes aplicativos, diferentes sintaxes.
Fonte: JARGAS, 2004, p. 13.

CASO

O Unix é um sistema operativo portável e multitarefa, desenvolvido nos laboratórios da AT&T. Sua história teve início em meados da década de 1960. Na década de 1970, aderiu a um modelo de segurança baseado em permissões de acesso, usando como formato usuário-grupo-outros. Esse modelo foi adotado na criação do Linux e, inclusive, é usado até hoje nos sistemas GNU /Linux. Esses mecanismos baseados em permissões não foram seguros o suficiente para bloquear alguns tipos de ataques ao sistema operacional e aplicativos em uso, como, por exemplo, o "Ataque do Dia 03" e "Buffer Overflow4" (RED, 2005, apud FERREIRA; TOLEDO, 2013, p. 2). Com o propósito de sofisticar o seu sistema de segurança, a National Security Agency iniciou um projeto de implementação de diversas modificações no kernel do Linux. Assim, surgiu o projeto "Security-Enhanced Linux" (Segurança Aprimorada Linux), pontualmente conhecido como SELinux (FERREIRA; TOLEDO, 2013).

Na sociedade da informação e do conhecimento, como você já sabe, os dados são os maiores bens de uma empresa e devem ser protegidos. A segurança da informação, portanto, é imprescindível!

VAMOS PRATICAR?

Expressões regulares são utilizadas para, por exemplo, procurar sequências de caracteres dentro de um arquivo. Assim, crie um arquivo textual contendo vários números de CPF (alguns válidos e outros inválidos) e depois uma Expressão Regular para verificar se os CPFs estão dentro do formato válido.

4.4 Função primária versus controle de conteúdo

Na década de 1990, a globalização foi intensificada e seus efeitos sociopolíticos e culturais também. Há uma evolução constante, onde a tecnologia prevalece. O estilo de vida das pessoas traz mudanças nunca antes vistas na obtenção de informações, facilitando sua criação, distribuição e manipulação. A revolução das telecomunicações está viabilizando o acesso dos usuários às informações em qualquer lugar e a qualquer hora.
Essa tendência, representada pela computação ubíqua, cria uma necessidade de as organizações estarem interconectadas a sistemas heterogêneos. Dessa forma, a administração da segurança desses ambientes heterogêneos torna-se desafiadora e complexa, tendo em vista as distintas características de segurança de cada ambiente envolvido.

4.4.1 Sistemas operacionais de rede

Dentre os sistemas operacionais de rede mais utilizados atualmente, podemos citar os sistemas Unix e seus derivados, como o GNU/Linux, o FreeBSD etc. (FERREIRA; TOLEDO, 2013).
Um sistema operacional de rede é dotado de mecanismos que permitem o gerenciamento de uma rede de computadores. Tal gerenciamento é associado aos serviços tanto do lado dos clientes quanto do lado dos servidores de uma rede. Dentre os serviços básicos, estão os relacionados ao servidor de arquivos, servidor de banco de dados, servidor de endereços IP (*Internet Protocol*, ou protocolo de Internet) e os servidores de comunicação propriamente ditos.

VAMOS PRATICAR?

Faça um levantamento dos sistemas operacionais de rede mais utilizados atualmente no mercado e verifique a possibilidade de aplicar programação Shell Script em suas configurações básicas. Em tal análise, verifique também em quais itens normalmente utiliza-se Shell Script para configurações e administração.

Ao Linux, saiba que poderão ser adicionadas e ativadas várias funcionalidades relacionadas ao contexto de rede. Dentre essas funcionalidades, podemos destacar: protocolo DHCP (*Dynamic Host Configuration Protocol*, ou

Protocolo de Configuração Dinâmica de Host); serviços de roteamento; protocolo DNS (*Domain Name Server*, ou Servidor de Nomes de Domínio); e serviços de cooperação com o sistema operacional Windows (NEMETH; SNYDER; HEIN, 2007).

Com as atividades propostas, você terá a oportunidade de fixar o conteúdo e fortalecer sua memória, além de consolidar hábitos de disciplina importantes para sua vida acadêmica.

Síntese

Nesta unidade, você aprendeu que a programação em Shell permite aos administradores criar pequenos programas para automatizar a administração de um sistema. Programar em Shell significa criar um arquivo de texto contendo comandos do Unix e comandos especiais do Shell para variáveis, testes, laços, funções e comentários. Os comandos podem ser enviados de duas maneiras para o interpretador: de forma interativa e não interativa e interativa. Você também aprendeu sobre o editor Vi, funções para o encadeamento de rotina Shell, comandos e variáveis internas, entre outros detalhes pertinentes ao assunto.

Nesta unidade, você teve a oportunidade de:

- conhecer os conceitos fundamentais sobre blocos de comandos e sub-rotinas de programação;
- demonstrar e analisar o funcionamento de scripts em Shell que utilizam funções declaradas;
- conhecer as características essenciais de um editor de texto a ser utilizado por desenvolvedores de scripts Shell.

Bibliografia

COELHO, R. S. A. **Editor Vi**. São Paulo: Novatec, 2002.

COSTA, D. G. **Administração de redes com scripts**: Bash Script, Python e VBScript. 2. ed. Rio de Janeiro: Brasport, 2010.

FERREIRA, F. A.; TOLEDO, A. S. O. **SElinux**: como tornar um servidor mais seguro. Disponível em: http://blog.newtonpaiva.br/pos/wp-content/uploads/2013/02/E3-SI-21.pdf. Acesso em: 30/09/2019.

JARGAS, A. M. **Conhecendo as Expressões Regulares**. 2004. Disponível em: http://aurelio.net/regex/apostila-conhecendo-regex.pdf. Acesso em: 30/09/2019.

NEMETH, E.; SNYDER, G.; HEIN, T. R. **Manual Completo do LINUX**: Guia do Administrador. São Paulo: Pearson Prentice Hall, 2007.

PET-TELE UFF. **Introdução ao Linux e programação em Shell Script**. Programa de Educação Tutorial Telecomunicações. Niterói - RJ, 2004. Disponível em: https://www.telecom.uff.br/pet/petws/downloads/apostilas/LINUX.pdf. Acesso em: 30/09/2019.

SANTOS, G. Comandos básicos de Shell Script (como criar, dar permissões de execução, ler input etc.). Publicado em 28/05/2019. Disponível em: https://www.youtube.com/watch?v=zZXxpXnxLj4. Acesso em 08/10/2019.

SHELL Script do zero. Disponível em: http://metamorphoselinux.net/Shell_Script_do_Zero.pdf. Acesso em 05/10/2019.