103.7 Lição 1

Certificação:	LPIC-1
Versão:	5.0
Tópico:	103 Comandos GNU e Unix
Objetivo:	103.7 Pesquisar em arquivos de texto usando expressões regulares
Lição:	1 de 2

Introdução

Os algoritmos de pesquisa de string são muito usados por diversas tarefas de processamento de dados, a tal ponto que os sistemas operacionais Unix têm sua própria implementação onipresente: as *expressões regulares*, freqüentemente abreviadas para *ERs*. As expressões regulares consistem em sequências de caracteres constituindo um padrão genérico usado para localizar e, às vezes, modificar uma sequência correspondente em uma sequência maior de caracteres. As expressões regulares expandem muito a capacidade de:

- Escrever regras de análise (parsing) para solicitações em servidores HTTP, *nginx* em particular.
- Escrever scripts que convertem conjuntos de dados baseados em texto para outro formato.
- Procurar por ocorrências de interesse em entradas de diário ou documentos.
- Filtrar documentos de marcação, mantendo o conteúdo semântico.

A expressão regular mais simples contém pelo menos um *átomo*. Um átomo, assim chamado por ser o elemento básico de uma expressão regular, é apenas um caractere que pode ou não ter um significado especial. A maioria dos caracteres comuns não são ambíguos e retêm seu significado

literal, enquanto outros têm um significado especial:

`.` (ponto)
O átomo corresponde a qualquer caractere.

`^` (acento circunflexo)
O átomo corresponde ao início de uma linha.

`$` (cifrão)
O átomo corresponde ao fim de uma linha.

Por exemplo, a expressão regular `bc`, composta pelos átomos literais `b` e `c`, pode ser encontrada na string `abcd`, mas não na string `a1cd`. Por outro lado, a expressão regular `.c` pode ser encontrada nas strings `abcd` e `a1cd`, pois o ponto `.` corresponde a qualquer caractere.

Os átomos de circunflexo e o cifrão são usados quando apenas as correspondências no início ou no final da string são de interesse. Por essa razão, eles também são chamados de *âncoras*. Assim, `cd` pode ser encontrado em `abcd`, mas `^cd` não. Da mesma forma, `ab` pode ser encontrado em `abcd`, mas `ab$` não. O acento circunflexo `^` é um caractere literal, exceto quando no início, e `$` é um caractere literal, exceto quando no final da expressão regular.

Expressão de colchetes

Existe outro tipo de átomo denominado *expressão de colchetes*. Embora não sejam um único caractere, os colchetes `[]` (incluindo o conteúdo) são considerados um único átomo. Uma expressão de colchetes geralmente é apenas uma lista de caracteres literais delimitados por `[]`, fazendo com que o átomo corresponda a qualquer caractere único da lista. Por exemplo, a expressão `[1b]` pode ser encontrada nas strings `abcd` e `a1cd`. Para especificar caracteres aos quais o átomo não deve corresponder, a lista deve começar com `^`, como em `[^1b]`. Também é possível especificar intervalos de caracteres em expressões de colchetes. Por exemplo, `[0-9]` corresponde aos dígitos de 0 a 9 e `[a-z]` corresponde a qualquer letra minúscula. Os intervalos devem ser usados com cuidado, pois podem não ser consistentes em diferentes idiomas.

As listas de expressão de colchetes também aceitam classes em vez de apenas caracteres únicos e intervalos. As classes de caracteres tradicionais são:

`[:alnum:]`
Representa um caractere alfanumérico.

`[:alpha:]`
Representa um caractere alfabético.

`[:ascii:]`

Representa um caractere que pertence ao conjunto de caracteres ASCII.

`[:blank:]`

Representa um caractere em branco, ou seja, um espaço ou tabulação.

`[:cntrl:]`

Representa um caractere de controle.

`[:digit:]`

Representa um dígito (de 0 a 9).

`[:graph:]`

Representa qualquer caractere imprimível, exceto espaço.

`[:lower:]`

Representa um caractere em minúsculas.

`[:print:]`

Representa qualquer caractere imprimível, incluindo espaço.

`[:punct:]`

Representa qualquer caractere imprimível que não seja um espaço ou um caractere alfanumérico.

`[:space:]`

Representa os caracteres de espaço em branco: espaço, alimentação de formulário (`\f`), nova linha (`\n`), retorno de carro (`\r`), tabulação horizontal (`\t`) e tabulação vertical (`\v`).

`[:upper:]`

Representa uma letra maiúscula.

`[:xdigit:]`

Representa dígitos hexadecimais (de 0 a F).

As classes de caracteres podem ser combinadas com caracteres únicos e intervalos, mas não podem ser usadas como ponto final de um intervalo. Além disso, as classes de caracteres podem ser usadas apenas em expressões de colchetes, não como um átomo independente fora dos colchetes.

Quantificadores

O alcance de um átomo, seja um átomo de caractere único ou um átomo de colchete, pode ser ajustado usando um *quantificador de átomo*. Os quantificadores de átomos definem *seqüências* de átomos, ou seja, as correspondências ocorrem quando uma repetição contígua para o átomo é encontrada na string. A substring que casa com a correspondência é chamada de *peça*. Não obstante, quantificadores e outros recursos de expressões regulares são tratados de maneira diferente dependendo do padrão que está sendo usado.

Conforme definido pelo POSIX, existem duas formas de expressões regulares: expressões regulares "básicas" e expressões regulares "estendidas". A maioria dos programas que trabalham com texto, em qualquer distribuição Linux convencional, oferece suporte a ambas as formas, por isso é importante conhecer as diferenças para evitar problemas de compatibilidade e escolher a implementação mais adequada para a tarefa em questão.

O quantificador `*` tem a mesma função em ERs básicas e estendidas (o átomo ocorre zero ou mais vezes) e é um caractere literal se aparecer no início da expressão regular ou se for precedido por uma barra invertida `\`. O quantificador de sinal de mais `+` seleciona as peças que contêm uma ou mais correspondências de átomos em sequência. Com o quantificador de ponto de interrogação `?`, a correspondência ocorrerá se o átomo correspondente aparecer uma vez ou se não aparecer. Se precedido por uma barra invertida `\`, seu significado especial não é considerado. As expressões regulares básicas também suportam os quantificadores `+` e `?`, mas eles precisam ser precedidos por uma barra invertida. Ao contrário das expressões regulares estendidas, `+` e `?` sozinhos são caracteres literais em expressões regulares básicas.

Chaves

As *chaves* são quantificadores que permitem ao usuário especificar limites precisos de quantidade para um átomo. Em expressões regulares estendidas, as chaves podem aparecer de três maneiras:

`{i}`

O átomo deve aparecer exatamente `i` vezes (sendo `i` um número inteiro). Por exemplo, `[[:blank:]]{2}` corresponde a exatamente dois caracteres em branco.

`{i,}`

O átomo deve aparecer pelo menos `i` vezes (sendo `i` um número inteiro). Por exemplo,`[[:blank:]]{2,}` corresponde a qualquer sequência de dois ou mais caracteres em branco.

`{i,j}`

The atom must appear at least `i` times and at most `j` times (`i` and `j` integer numbers, `j` greater

then `i`). For example, `xyz{2,4}` matches the `xy` string followed by two to four of the `z` character.

De toda forma, se uma substring corresponder a uma expressão regular e uma substring mais longa começando no mesmo ponto também corresponder, a substring mais longa será considerada.

As expressões regulares básicas também suportam chaves, mas elas devem ser precedidas por `\`: `\{` e `\}`. Sozinhas, `{` e `}` são interpretadas como caracteres literais. Uma `\{` seguida por um caractere diferente de um dígito é um caractere literal, não uma abertura de chave.

Alternâncias e agrupamentos

As expressões regulares básicas também diferem das expressões regulares estendidas em outro aspecto importante: uma expressão regular estendida pode ser dividida em *alternâncias* (branches), sendo cada uma delas uma expressão regular independente. As alternativas são separadas por `|` e a expressão regular combinada corresponderá a qualquer coisa que corresponda a qualquer uma das alternativas. Por exemplo, `he|him` irá corresponder se a substring `he` ou a substring `him` forem encontradas na string que está sendo examinada. As expressões regulares básicas interpretam `|` como um caractere literal. No entanto, a maioria dos programas que suportam expressões regulares básicas permitem alternâncias com `\|`.

Uma expressão regular estendida entre `()` pode ser usada em um *agrupamento* (back reference). Por exemplo, `([[:digit:]])\1` corresponde a qualquer expressão regular que se repita pelo menos uma vez, porque o `\1` na expressão é o agrupamento da peça encontrada pela primeira subexpressão entre parênteses. Se houver mais de uma subexpressão entre parênteses na expressão regular, elas podem ser referenciadas com `\2`, `\3` e assim por diante.

Para ERs básicas, as subexpressões devem ser colocadas entre `\(` e `\)`, sendo `(` and `)` sozinhos caracteres comuns. O indicador de agrupamento é usado como nas expressões regulares estendidas.

Pesquisas com expressões regulares

A vantagem imediata oferecida pelas expressões regulares é aprimorar as pesquisas em sistemas de arquivos e em documentos de texto. A opção `-regex` do comando `find` permite testar cada caminho em uma hierarquia de diretórios de acordo com uma expressão regular. Por exemplo,

```
$ find $HOME -regex '.*/\..*' -size +100M
```

busca por arquivos maiores que 100 megabytes (100 unidades de 1048576 bytes), mas apenas em caminhos dentro do diretório inicial do usuário que contenham uma correspondência com `.*/\..*`, ou seja, um `/.` rodeado por qualquer outro número de caracteres. Em outras palavras, apenas os arquivos ocultos ou arquivos dentro de diretórios ocultos serão listados, independentemente da

posição de `/.` no caminho correspondente. Para expressões regulares que não diferenciam maiúsculas de minúsculas, a opção `-iregex` deve ser usada em seu lugar:

```
$ find /usr/share/fonts -regextype posix-extended -iregex
'.*(dejavu|liberation).*sans.*(italic|oblique).*'
/usr/share/fonts/dejavu/DejaVuSansCondensed-BoldOblique.ttf
/usr/share/fonts/dejavu/DejaVuSansCondensed-Oblique.ttf
/usr/share/fonts/dejavu/DejaVuSans-BoldOblique.ttf
/usr/share/fonts/dejavu/DejaVuSans-Oblique.ttf
/usr/share/fonts/dejavu/DejaVuSansMono-BoldOblique.ttf
/usr/share/fonts/dejavu/DejaVuSansMono-Oblique.ttf
/usr/share/fonts/liberation/LiberationSans-BoldItalic.ttf
/usr/share/fonts/liberation/LiberationSans-Italic.ttf
```

Neste exemplo, a expressão regular contém alternâncias (escritas no estilo *estendido*) para listar apenas arquivos de fonte específicos na hierarquia de diretório `/usr/share/fonts`. Expressões regulares estendidas não são suportadas por padrão, mas `find` permite que sejam habilitadas com `-regextype posix-extended` or `-regextype egrep`. A ER padrão para `find` é *findutils-default*, que é essencialmente um clone da expressão regular básica.

Freqüentemente, é necessário passar a saída de um programa para o comando `less` quando ele não cabe na tela. O comando `less` divide a entrada em páginas, uma tela de cada vez, permitindo ao usuário navegar facilmente pelo texto para cima e para baixo. Além disso, `less` também permite que um usuário execute pesquisas baseadas em expressões regulares. Este recurso é notavelmente importante porque `less` é o paginador padrão usado para muitas tarefas diárias, como inspecionar entradas de diário ou consultar páginas de manual. Ao ler uma página de manual, por exemplo, pressionar a tecla `/` abre um prompt de pesquisa. Esse é um cenário típico em que as expressões regulares são úteis, pois as opções do comando são listadas logo após a margem da página no layout geral da página do manual. No entanto, a mesma opção pode aparecer muitas vezes no texto, inviabilizando as buscas literais. Independentemente disso, digitar `^[[:blank:]]*-o` — ou, mais simplesmente: `^ *-o` — no prompt de pesquisa permite pular imediatamente para a opção da seção `-o` (se existente) após pressionar Enter, permitindo assim consultar mais rapidamente uma descrição da opção.

Exercícios Guiados

1. Qual expressão regular estendida corresponderia a qualquer endereço de e-mail, como `info@example.org`?

2. Qual expressão regular estendida corresponderia apenas a qualquer endereço IPv4 no formato pontilhado-quad padrão, como `192.168.15.1`?

3. Como o comando `grep` pode ser usado para listar o conteúdo do arquivo `/etc/services`, descartando todos os comentários (linhas começando com `#`)?

4. O arquivo `domains.txt` contém uma lista de nomes de domínio, um por linha. Como o comando `egrep` seria usado para listar apenas domínios `.org` or `.com`?

Exercícios Exploratórios

1. A partir do diretório atual, como o comando `find` usaria uma expressão regular estendida para pesquisar todos os arquivos que não contêm um sufixo de arquivo padrão (nomes de arquivo que não terminam em `.txt` ou `.c`, por exemplo)?

2. O comando `less` é o paginador padrão para exibir arquivos de texto longos no ambiente shell. Ao digitar `/`, uma expressão regular pode ser inserida no prompt de pesquisa para pular para a primeira correspondência pertinente. Para permanecer na posição atual do documento e destacar apenas as correspondências pertinentes, qual combinação de teclas deve ser inserida no prompt de pesquisa?

3. Em `less`, como seria possível filtrar a saída para que apenas as linhas que correspondem a uma expressão regular sejam exibidas?

Resumo

Esta lição cobre o suporte geral do Linux para expressões regulares, um padrão amplamente usado cujos recursos de correspondência de padrões são suportados pela maioria dos programas que trabalham com texto. A lição atravessa as seguintes etapas:

- O que é uma expressão regular.
- Os principais componentes de uma expressão regular.
- As diferenças entre expressões regulares e expressões regulares estendidas.
- Como realizar pesquisas simples de texto e arquivos usando expressões regulares.

Respostas aos Exercícios Guiados

1. Qual expressão regular estendida corresponderia a qualquer endereço de e-mail, como `info@example.org`?

```
egrep "\S+@\S+\.\S+"
```

2. Qual expressão regular estendida corresponderia apenas a qualquer endereço IPv4 no formato pontilhado-quad padrão, como `192.168.15.1`?

```
egrep "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}"
```

3. Como o comando `grep` pode ser usado para listar o conteúdo do arquivo `/etc/services`, descartando todos os comentários (linhas começando com `#`)?

```
grep -v ^# /etc/services
```

4. O arquivo `domains.txt` contém uma lista de nomes de domínio, um por linha. Como o comando `egrep` seria usado para listar apenas domínios `.org` or `.com`?

```
egrep ".org$|.com$" domains.txt
```

Respostas aos Exercícios Exploratórios

1. A partir do diretório atual, como o comando `find` usaria uma expressão regular estendida para pesquisar todos os arquivos que não contêm um sufixo de arquivo padrão (nomes de arquivo que não terminam em `.txt` ou `.c`, por exemplo)?

   ```
   find . -type f -regextype egrep -not -regex '.*\.[[:alnum:]]{1,}$'
   ```

2. O comando `less` é o paginador padrão para exibir arquivos de texto longos no ambiente shell. Ao digitar `/`, uma expressão regular pode ser inserida no prompt de pesquisa para pular para a primeira correspondência pertinente. Para permanecer na posição atual do documento e destacar apenas as correspondências pertinentes, qual combinação de teclas deve ser inserida no prompt de pesquisa?

   Pressionando `Ctrl` + `K` antes de inserir a expressão de pesquisa.

3. Em `less`, como seria possível filtrar a saída para que apenas as linhas que correspondem a uma expressão regular sejam exibidas?

   Pressionando `&` e inserindo a expressão de pesquisa.

103.7 Lição 2

Certificação:	LPIC-1
Versão:	5.0
Tópico:	103 Comandos GNU e Unix
Objetivo:	103.7 Pesquisar em arquivos de texto usando expressões regulares
Lição:	2 de 2

Introdução

O streaming de dados por meio de uma cadeia de comandos canalizados permite a aplicação de filtros compostos com base em expressões regulares. As expressões regulares são uma técnica importante usada não apenas na administração de sistemas, mas também na *mineração de dados* e áreas relacionadas. Dois comandos são especialmente adequados para manipular arquivos e texto usando expressões regulares: `grep` e `sed`. `grep` é um localizador de padrões e `sed` é um editor de fluxo. Eles são úteis por si sós, mas é no trabalho em conjunto com outros processos que se destacam de fato.

O localizador de padrões: grep

Um dos usos mais comuns do `grep` é facilitar a inspeção de arquivos longos, usando a expressão regular como filtro aplicado a cada linha. Pode ser usado para mostrar apenas as linhas que começam com um determinado termo. Por exemplo, o `grep` pode ser usado para investigar um arquivo de configuração para módulos do kernel, listando apenas as linhas de opções:

```
$ grep '^options' /etc/modprobe.d/alsa-base.conf
```

```
options snd-pcsp index=-2
options snd-usb-audio index=-2
options bt87x index=-2
options cx88_alsa index=-2
options snd-atiixp-modem index=-2
options snd-intel8x0m index=-2
options snd-via82xx-modem index=-2
```

O caractere pipe `|` pode ser empregado para redirecionar a saída de um comando diretamente para a entrada de `grep`. O exemplo a seguir usa uma expressão entre colchetes para selecionar linhas da saída de `fdisk -l`, começando com `Disk /dev/sda` ou `Disk /dev/sdb`:

```
# fdisk -l | grep '^Disk /dev/sd[ab]'
Disk /dev/sda: 320.1 GB, 320072933376 bytes, 625142448 sectors
Disk /dev/sdb: 7998 MB, 7998537728 bytes, 15622144 sectors
```

A mera seleção de linhas com correspondências pode não ser apropriada para uma tarefa em particular, exigindo ajustes no comportamento de `grep` através de suas opções. Por exemplo, a opção `-c` ou `--count` diz ao `grep` para exibir quantas linhas têm correspondências:

```
# fdisk -l | grep '^Disk /dev/sd[ab]' -c
2
```

A opção pode ser colocada antes ou depois da expressão regular. Outras opções importantes do `grep` são:

`-c` ou `--count`

Em vez de exibir os resultados da pesquisa, exibe apenas a contagem total de quantas vezes uma correspondência ocorre em qualquer arquivo.

`-i` ou `--ignore-case`

Torna a busca insensível a maiúsculas e minúsculas.

`-f FILE` ou `--file=FILE`

Indica um arquivo contendo a expressão regular a ser usada.

`-n` ou `--line-number`

Mostra o número da linha.

`-v` ou `--invert-match`

Seleciona todas as linhas, exceto aquelas que contêm correspondências.

`-H` ou `--with-filename`

Mostra também o nome do arquivo que contém a linha.

`-z` ou `--null-data`

Em vez de fazer com que o `grep` trate os fluxos de dados de entrada e saída como linhas separadas (usando *newline* por padrão), ele passa a encarar a entrada ou saída como uma sequência de linhas. Ao combinar a saída do comando `find` usando a opção `-print0` com o comando `grep`, a opção `-z` ou `--null-data` deve ser usada para processar o fluxo da mesma maneira.

Embora ativada por padrão quando vários caminhos de arquivo são fornecidos como entrada, a opção `-H` não é ativada para arquivos individuais. Isso pode ser importante em situações especiais, como quando `grep` é chamado diretamente por `find`, por exemplo:

```
$ find /usr/share/doc -type f -exec grep -i '3d modeling' "{}" \; | cut -c -100
artistic aspects of 3D modeling. Thus this might be the application you are
This major approach of 3D modeling has not been supported
oce is a C++ 3D modeling library. It can be used to develop CAD/CAM softwares, for
instance [FreeCad
```

Neste exemplo, `find` lista todos os arquivos em `/usr/share/doc` e passa cada um deles para o `grep`, que por sua vez realiza uma busca sem distinção entre maiúsculas e minúsculas por `3d modeling` dentro do arquivo. O pipe para `cut` existe apenas para limitar o comprimento da saída a 100 colunas. Observe, entretanto, que não há como saber de qual arquivo as linhas vieram. Esse problema é resolvido adicionando `-H` ao `grep`:

```
$ find /usr/share/doc -type f -exec grep -i -H '3d modeling' "{}" \; | cut -c -100
/usr/share/doc/openscad/README.md:artistic aspects of 3D modeling. Thus this might
be the applicatio
/usr/share/doc/opencsg/doc/publications.html:This major approach of 3D modeling has
not been support
```

Agora é possível identificar os arquivos nos quais cada correspondência foi encontrada. Para tornar a listagem ainda mais informativa, linhas iniciais e finais podem ser adicionadas às linhas com correspondências:

```
$ find /usr/share/doc -type f -exec grep -i -H -1 '3d modeling' "{}" \; | cut -c
```

```
-100
/usr/share/doc/openscad/README.md-application Blender), OpenSCAD focuses on the CAD
aspects rather t
/usr/share/doc/openscad/README.md:artistic aspects of 3D modeling. Thus this might
be the applicatio
/usr/share/doc/openscad/README.md-looking for when you are planning to create 3D
models of machine p
/usr/share/doc/opencsg/doc/publications.html-3D graphics library for Constructive
Solid Geometry (CS
/usr/share/doc/opencsg/doc/publications.html:This major approach of 3D modeling has
not been support
/usr/share/doc/opencsg/doc/publications.html-by real-time computer graphics until
recently.
```

A opção `-1` instrui o `grep` a incluir uma linha antes e uma linha depois quando encontrar uma linha com uma correspondência. Essas linhas extras são chamadas de *linhas de contexto* e são identificadas na saída por um sinal de menos após o nome do arquivo. O mesmo resultado pode ser obtido com `-C 1` ou `--context=1`; outras quantidades de linhas de contexto podem ser indicadas.

Existem dois programas complementares a grep: `egrep` e `fgrep`. O programa `egrep` é equivalente ao comando `grep -E`, que incorpora recursos extras além das expressões regulares básicas. Por exemplo, com o `egrep` é possível usar recursos de expressões regulares estendidas, como as alternâncias:

```
$ find /usr/share/doc -type f -exec egrep -i -H -1 '3d (modeling|printing)' "{}" \;
| cut -c -100
/usr/share/doc/openscad/README.md-application Blender), OpenSCAD focuses on the CAD
aspects rather t
/usr/share/doc/openscad/README.md:artistic aspects of 3D modeling. Thus this might
be the applicatio
/usr/share/doc/openscad/README.md-looking for when you are planning to create 3D
models of machine p
/usr/share/doc/openscad/RELEASE_NOTES.md-* Support for using 3D-Mouse / Joystick /
Gamepad input dev
/usr/share/doc/openscad/RELEASE_NOTES.md:* 3D Printing support: Purchase from a
print service partne
/usr/share/doc/openscad/RELEASE_NOTES.md-* New export file formats: SVG, 3MF, AMF
/usr/share/doc/opencsg/doc/publications.html-3D graphics library for Constructive
Solid Geometry (CS
/usr/share/doc/opencsg/doc/publications.html:This major approach of 3D modeling has
not been support
/usr/share/doc/opencsg/doc/publications.html-by real-time computer graphics until
recently.
```

Neste exemplo, a expressão buscada corresponderá a `3D modeling` ou `3D printing`, sem distinção entre maiúsculas e minúsculas. Para exibir apenas as partes de um fluxo de texto que correspondem à expressão usada por `egrep`, use a opção `-o`.

O programa `fgrep` é equivalente a `grep -F`, ou seja, não analisa expressões regulares. Ele é útil em pesquisas simples, nas quais o objetivo é encontrar uma expressão literal. Portanto, caracteres especiais como o cifrão e o ponto serão interpretados literalmente e não como seus significados em uma expressão regular.

O editor de fluxo: sed

O objetivo do programa `sed` é modificar dados baseados em texto de forma não interativa. Isso significa que toda a edição é feita por instruções predefinidas, não por uma digitação direta e arbitrária em um texto exibido na tela. Em termos modernos, o `sed` pode ser entendido como um analisador de modelo: dado um texto como entrada, ele coloca um conteúdo personalizado em posições predefinidas ou quando encontra uma correspondência para uma expressão regular.

O sed é adequado para texto transmitido por meio de encadeamentos (pipelines). Sua sintaxe básica é `sed -f SCRIPT`, quando as instruções de edição são armazenadas no arquivo `SCRIPT`, ou `sed -e COMANDOS` para executar `COMANDOS` diretamente da linha de comando. Se nem `-f` nem `-e` estiverem presentes, o `sed` usa o primeiro parâmetro que não seja uma opção como arquivo de script. Também é possível usar um arquivo como entrada apenas fornecendo seu caminho como um argumento para o `sed`.

As instruções do `sed` são compostas por um único caractere, possivelmente precedido por um endereço ou seguido por uma ou mais opções, e são aplicadas a uma linha de cada vez. Os endereços podem ser um único número de linha, uma expressão regular ou um intervalo de linhas. Por exemplo, a primeira linha de um fluxo de texto pode ser excluída com `1d`, onde `1` especifica a linha à qual o comando de exclusão `d` será aplicado. Para esclarecer o uso de `sed`, vejamos a saída do comando `factor `seq 12``, que retorna os fatores primos para os números de 1 a 12:

```
$ factor `seq 12`
1:
2: 2
3: 3
4: 2 2
5: 5
6: 2 3
7: 7
8: 2 2 2
9: 3 3
10: 2 5
```

```
11: 11
12: 2 2 3
```

A exclusão da primeira linha com o `sed` é realizada por `1d`:

```
$ factor `seq 12` | sed 1d
2: 2
3: 3
4: 2 2
5: 5
6: 2 3
7: 7
8: 2 2 2
9: 3 3
10: 2 5
11: 11
12: 2 2 3
```

Especificamos um intervalo de linhas com uma vírgula de separação:

```
$ factor `seq 12` | sed 1,7d
8: 2 2 2
9: 3 3
10: 2 5
11: 11
12: 2 2 3
```

Mais de uma instrução pode ser usada na mesma execução, separadas por ponto e vírgula. Nesse caso, no entanto, é importante colocá-las entre parênteses para que o ponto e vírgula não seja interpretado pelo shell:

```
$ factor `seq 12` | sed "1,7d;11d"
8: 2 2 2
9: 3 3
10: 2 5
12: 2 2 3
```

Neste exemplo, duas instruções de exclusão foram executadas, primeiro nas linhas que vão de 1 a 7 e depois na linha 11. Um endereço também pode ser uma expressão regular, portanto, apenas as linhas correspondentes serão afetadas pela instrução:

```
$ factor `seq 12` | sed "1d;/:.*2.*/d"
3: 3
5: 5
7: 7
9: 3 3
11: 11
```

A expressão regular `:.*2.*` corresponde a qualquer ocorrência do número 2 em qualquer lugar depois de dois pontos, provocando a exclusão das linhas correspondentes aos números que tenham 2 como fator. Com o `sed`, qualquer coisa colocada entre barras (/) é considerada uma expressão regular e, por padrão, todas as ER básicas são suportadas. Por exemplo, `sed -e "/^#/d" /etc/services` mostra o conteúdo do arquivo `/etc/services` sem as linhas que começam com `#` (linhas de comentário).

A instrução de exclusão `d` é apenas uma das muitas instruções de edição fornecidas pelo `sed`. Em vez de excluir uma linha, o `sed` pode substituí-la por um texto determinado:

```
$ factor `seq 12` | sed "1d;/:.*2.*/c REMOVED"
REMOVED
3: 3
REMOVED
5: 5
REMOVED
7: 7
REMOVED
9: 3 3
REMOVED
11: 11
REMOVED
```

A instrução `c REMOVED` simplesmente substitui uma linha pelo texto `REMOVED`. No caso do exemplo, cada linha com uma substring correspondente à expressão regular `:.*2.*` é afetada pela instrução `c REMOVED`. A instrução `a TEXT` copia o texto indicado por `TEXT` para uma nova linha após a linha com uma correspondência. A instrução `r FILE` faz o mesmo, mas copia o conteúdo do arquivo indicado por `FILE`. A instrução `w` faz o oposto de `r`, ou seja, a linha será anexada ao arquivo indicado.

De longe, a instrução mais usada do `sed` é `s/FIND/REPLACE/`, que é usada para substituir uma correspondência da expressão regular `FIND` pelo texto indicado por `REPLACE`. Por exemplo, a instrução `s/hda/sda/` substitui uma substring que corresponde à ER literal `hda` por `sda`. Apenas a primeira correspondência encontrada na linha será substituída, a menos que o sinalizador `g` seja colocado após a instrução, como em `s/hda/sda/g`.

Um estudo de caso mais realístico ajudará a ilustrar os recursos do `sed`. Suponha que uma clínica médica queira enviar mensagens de texto a seus clientes, lembrando-os de suas consultas agendadas para o dia seguinte. Um cenário de implementação típico depende de um serviço profissional de mensagens instantâneas, que fornece uma API para acessar o sistema responsável pela entrega das mensagens. Essas mensagens geralmente são originadas do mesmo sistema que executa o aplicativo de controle de agendamentos do cliente, acionadas por um horário específico do dia ou algum outro evento. Nessa situação hipotética, o aplicativo poderia gerar um arquivo chamado `appointments.csv` contendo dados tabulados com todas as consultas agendadas para o dia seguinte, que seria usado pelo `sed` para construir as mensagens de texto a partir de um arquivo de modelo chamado `template.txt` . Os arquivos CSV são uma forma padrão de exportar dados de agendamentos de banco de dados. Portanto, os dados padrão dos agendamentos poderiam ser fornecidos da seguinte forma:

```
$ cat appointments.csv
"NAME","TIME","PHONE"
"Carol","11am","55557777"
"Dave","2pm","33334444"
```

A primeira linha contém os rótulos de cada coluna, que serão usados para preencher as tags dentro do arquivo de modelo:

```
$ cat template.txt
Hey <NAME>, don't forget your appointment tomorrow at <TIME>.
```

Os sinais de menor que `<` e maior que `>` foram postos em torno dos rótulos apenas para ajudar a identificá-los como tags. O script Bash a seguir analisa todos os agendamentos da fila usando `template.txt` como modelo de mensagem:

```
#! /bin/bash

TEMPLATE=`cat template.txt`
TAGS=(`sed -ne '1s/^"//;1s/","/\n/g;1s/"$//p' appointments.csv`)
mapfile -t -s 1 ROWS < appointments.csv
for (( r = 0; r < ${#ROWS[*]}; r++ ))
do
  MSG=$TEMPLATE
  VALS=(`sed -e 's/^"//;s/","/\n/g;s/"$//' <<<${ROWS[$r]}`)
  for (( c = 0; c < ${#TAGS[*]}; c++ ))
  do
    MSG=`sed -e "s/<${TAGS[$c]}>/${VALS[$c]}/g" <<<"$MSG"`
```

```
  done
  echo curl --data message=\"$MSG\" --data phone=\"${VALS[2]}\"
https://mysmsprovider/api
done
```

Um script de produção real também cuidaria da autenticação, verificação de erros e registro, mas o exemplo mostra uma funcionalidade básica para começar. As primeiras instruções executadas pelo `sed` são aplicadas apenas à primeira linha — o endereço `1` em `1s/^"//;1s/","/\n/g;1s/"$//p` — para remover as aspas iniciais e finais — `1s/^"//` e `1s/"$//` — e substituir os separadores de campo por um caractere de nova linha: `1s/","/\n/g`. Apenas a primeira linha é necessária para carregar os nomes das colunas, de forma que as linhas não correspondentes serão suprimidas pela opção `-n`, exigindo que o sinalizador `p` seja colocado após o último comando `sed` para imprimir a linha correspondente. As tags são então armazenadas na variável `TAGS` como uma matriz Bash. Outra variável da matriz Bash é criada pelo comando `mapfile` para armazenar as linhas contendo os agendamentos na variável da matriz `ROWS`.

Um loop `for` é empregado para processar cada linha de agendamento encontrada em `ROWS`. Em seguida, as aspas e separadores no agendamento — o agendamento está na variável `${ROWS[$r]}` usada como uma *here string* — são substituídos por `sed`, como no caso dos comandos usados para carregar as tags. Os valores separados do agendamento são então armazenados na variável de matriz `VALS`, onde os subscritos de matriz 0, 1 e 2 correspondem aos valores de `NAME`, `TIME` e `PHONE`.

Finalmente, um loop `for` aninhado percorre a matriz `TAGS` e substitui cada tag encontrada no modelo por seu valor correspondente em `VALS`. A variável `MSG` contém uma cópia do modelo gerado, atualizado pelo comando de substituição `s/<${TAGS[$c]}>/${VALS[$c]}/g` em cada loop que passa por `TAGS`.

Isso resulta em uma mensagem semelhante a: `"Hey Carol, don't forget your appointment tomorrow at 11am."` (em português: `"Olá, Carol. Não se esqueça de sua consulta amanhã às 11h."`). A mensagem gerada pode então ser enviada como um parâmetro através de uma solicitação HTTP com `curl`, em forma de email ou qualquer outro método semelhante.

Combinando grep e sed

Os comandos `grep` e `sed` podem ser usados juntos quando uma mineração de texto mais complexa é necessária. Como administrador do sistema, você pode querer inspecionar todas as tentativas de login em um servidor, por exemplo. O arquivo `/var/log/wtmp` registra todos os logins e logouts, ao passo que `/var/log/btmp` registra as tentativas de login falhadas. Eles são escritos em um formato binário, que pode ser lido, respectivamente, pelos comandos `last` e `lastb`. A saída de `lastb` mostra não apenas o nome de usuário empregado na tentativa de login incorreta, mas também seu endereço IP:

```
# lastb -d -a -n 10 --time-format notime
user     ssh:notty       (00:00)     81.161.63.251
nrostagn ssh:notty       (00:00)     vmd60532.contaboserver.net
pi       ssh:notty       (00:00)     132.red-88-20-39.staticip.rima-tde.net
pi       ssh:notty       (00:00)     132.red-88-20-39.staticip.rima-tde.net
pi       ssh:notty       (00:00)     46.6.11.56
pi       ssh:notty       (00:00)     46.6.11.56
nps      ssh:notty       (00:00)     vmd60532.contaboserver.net
narmadan ssh:notty       (00:00)     vmd60532.contaboserver.net
nominati ssh:notty       (00:00)     vmd60532.contaboserver.net
nominati ssh:notty       (00:00)     vmd60532.contaboserver.net
```

A opção `-d` traduz o número IP para o nome de host correspondente. O nome do host pode fornecer pistas sobre o provedor de internet ou o serviço de hospedagem usado para realizar essas tentativas de login incorretas. A opção `-a` põe o nome do host na última coluna, o que facilita a filtragem a ser aplicada. A opção `--time-format notime` suprime a hora em que ocorreu a tentativa de login. O comando `lastb` pode levar algum tempo para ser concluído caso tenha havido muitas tentativas de login incorretas, por isso limitamos a saída a dez entradas com a opção `-n 10`. Nem todos os IPs remotos têm um nome de host associado, portanto o DNS reverso não se aplica e eles podem ser dispensados. Embora seja possível escrever uma expressão regular que corresponda ao formato esperado para um nome de host no final da linha, provavelmente é mais simples escrever uma que corresponda a uma letra do alfabeto ou a um único dígito no final linha. O exemplo a seguir mostra como o comando `grep` pega a listagem em sua entrada padrão e remove as linhas sem nomes de host:

```
# lastb -d -a --time-format notime | grep -v '[0-9]$' | head -n 10
nvidia   ssh:notty       (00:00)     vmd60532.contaboserver.net
n_tonson ssh:notty       (00:00)     vmd60532.contaboserver.net
nrostagn ssh:notty       (00:00)     vmd60532.contaboserver.net
pi       ssh:notty       (00:00)     132.red-88-20-39.staticip.rima-tde.net
pi       ssh:notty       (00:00)     132.red-88-20-39.staticip.rima-tde.net
nps      ssh:notty       (00:00)     vmd60532.contaboserver.net
narmadan ssh:notty       (00:00)     vmd60532.contaboserver.net
nominati ssh:notty       (00:00)     vmd60532.contaboserver.net
nominati ssh:notty       (00:00)     vmd60532.contaboserver.net
nominati ssh:notty       (00:00)     vmd60532.contaboserver.net
```

A opção `-v` do comando `grep` exibe apenas as linhas que não correspondem à expressão regular fornecida. Uma expressão regular que corresponda a qualquer linha terminada por um número (isto é, `[0-9]$`) captura apenas as entradas sem um nome de host. Portanto, `grep -v '[0-9]$'` mostra somente as linhas que terminam com um nome de host. A filtragem da saída pode ser ainda mais refinada, mantendo-se apenas o nome de domínio e removendo as outras partes de cada

`linha. Para isso, usamos o comando `sed` com um comando de substituição para trocar a linha inteira por um agrupamento referente ao nome de domínio contido nela:

```
# lastb -d -a --time-format notime | grep -v '[0-9]$' | sed -e 's/.* \(.*\)$/\1/' |
head -n 10
vmd60532.contaboserver.net
vmd60532.contaboserver.net
vmd60532.contaboserver.net
132.red-88-20-39.staticip.rima-tde.net
132.red-88-20-39.staticip.rima-tde.net
vmd60532.contaboserver.net
vmd60532.contaboserver.net
vmd60532.contaboserver.net
vmd60532.contaboserver.net
vmd60532.contaboserver.net
```

O parêntese escapado em `.* \(.*\)$` diz ao `sed` para lembrar aquela parte da linha, ou seja, a parte entre o último caractere de espaço e o final da linha. No exemplo, esta parte é referenciada com `\1` e usada para substituir a linha inteira. Parece claro que a maioria dos hosts remotos tenta fazer o login mais de uma vez; assim, o mesmo nome de domínio se repete. Para suprimir as entradas repetidas, elas precisam primeiro ser classificadas (com o comando `sort`) e depois passadas para o comando `uniq`:

```
# lastb -d -a --time-format notime | grep -v '[0-9]$' | sed -e 's/.* \(.*\)$/\1/' |
sort | uniq | head -n 10
116-25-254-113-on-nets.com
132.red-88-20-39.staticip.rima-tde.net
145-40-33-205.power-speed.at
tor.laquadrature.net
tor.momx.site
ua-83-226-233-154.bbcust.telenor.se
vmd38161.contaboserver.net
vmd60532.contaboserver.net
vmi488063.contaboserver.net
vmi515749.contaboserver.net
```

Este exemplo mostra como diferentes comandos podem ser combinados para produzir o resultado desejado. A lista de nomes de host pode então ser usada para escrever regras de firewall de bloqueio ou tomar outras medidas para garantir a segurança do servidor.

Exercícios Guiados

1. O comando `last` mostra uma lista dos últimos usuários logados, incluindo seus IPs de origem. Como o comando `egrep` poderia ser usado para filtrar a saída de `last`, mostrando apenas as ocorrências de um endereço IPv4, descartando quaisquer informações adicionais na linha correspondente?

2. Qual opção deve ser dada ao `grep` para filtrar corretamente a saída gerada pelo comando `find` executado com a opção `-print0`?

3. O comando `uptime -s` mostra a última data em que o sistema foi ligado, como em `2019-08-05 20:13:22`. Qual será o resultado do comando `uptime -s | sed -e 's/(.*) (.*)/\1/'`?

4. Qual opção deve ser dada ao `grep` para que ele conte as linhas correspondentes aos termos de pesquisa em vez de exibi-las?

Exercícios Exploratórios

1. A estrutura básica de um arquivo HTML começa com os elementos `html`, `head` e `body`, como por exemplo:

```
<html>
<head>
  <title>News Site</title>
</head>
<body>
  <h1>Headline</h1>
  <p>Information of interest.</p>
</body>
</html>
```

   Descreva como seria possível usar endereços no `sed` para exibir apenas o elemento `body` e seu conteúdo.

2. Qual expressão de `sed` poderia remover todas as tags de um documento HTML, mantendo apenas o texto?

3. Os arquivos com extensão `.ovpn` são muito populares para configurar clientes VPN, pois contêm não apenas as configurações, mas também o conteúdo de chaves e certificados para o cliente. Essas chaves e certificados estão originalmente em arquivos separados e portanto precisam ser copiados para o arquivo `.ovpn`. Dado o seguinte trecho de um modelo `.ovpn`:

```
client
dev tun
remote 192.168.1.155 1194
<ca>
ca.crt
</ca>
<cert>
client.crt
</cert>
<key>
client.key
</key>
<tls-auth>
```

```
ta.key
</tls-auth>
```

Supondo-se que os arquivos `ca.crt`, `client.crt`, `client.key` e `ta.key` estejam no diretório atual, como a configuração do modelo seria modificada pelo `sed` de forma a substituir cada nome de arquivo pelo seu conteúdo?

Resumo

Esta lição trata dos dois comandos mais importantes do Linux relacionados a expressões regulares: `grep` e `sed`. Os scripts e comandos compostos contam com `grep` e `sed` para realizar uma ampla gama de tarefas de filtragem e análise de texto. A lição inclui as seguintes etapas:

- Como usar o `grep` e suas variações, como `egrep` e `fgrep`.
- Como usar o `sed` e suas instruções internas para manipular texto.
- Exemplos de aplicações de expressões regulares usando `grep` e `sed`.

Respostas aos Exercícios Guiados

1. O comando `last` mostra uma lista dos últimos usuários logados, incluindo seus IPs de origem. Como o comando `egrep` poderia ser usado para filtrar a saída de `last`, mostrando apenas as ocorrências de um endereço IPv4, descartando quaisquer informações adicionais na linha correspondente?

   ```
   last -i | egrep -o '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}'
   ```

2. Qual opção deve ser dada ao `grep` para filtrar corretamente a saída gerada pelo comando `find` executado com a opção `-print0`?

   A opção `-z` or `--null-data`, como em `find . -print0 | grep -z expression`.

3. O comando `uptime -s` mostra a última data em que o sistema foi ligado, como em `2019-08-05 20:13:22`. Qual será o resultado do comando `uptime -s | sed -e 's/(.*) (.*)/\1/'`?

   Ocorrerá um erro. Por padrão, os parênteses devem ser escapados para ser possível usar agrupamentos no `sed`.

4. Qual opção deve ser dada ao `grep` para que ele conte as linhas correspondentes aos termos de pesquisa em vez de exibi-las?

   A opção `-c`.

Respostas aos Exercícios Exploratórios

1. A estrutura básica de um arquivo HTML começa com os elementos `html`, `head` e `body`, como por exemplo:

```
<html>
<head>
  <title>News Site</title>
</head>
<body>
  <h1>Headline</h1>
  <p>Information of interest.</p>
</body>
</html>
```

   Descreva como seria possível usar endereços no `sed` para exibir apenas o elemento `body` e seu conteúdo.

   Para mostrar apenas `body`, os endereços deveriam ser `/<body>/,/<\/body>/`, como em `sed -n -e '/<body>/,/<\/body>/p'`. A opção `-n` é dada ao `sed` para que ele não imprima linhas por padrão, por isso o comando `p` está no final da expressão de `sed` para imprimir as linhas correspondentes aos termos da pesquisa.

2. Qual expressão de `sed` poderia remover todas as tags de um documento HTML, mantendo apenas o texto?

   A expressão do `sed` `s/<[^>]*>//g` substitui qualquer conteúdo entre `<>` por uma string vazia.

3. Os arquivos com extensão `.ovpn` são muito populares para configurar clientes VPN, pois contêm não apenas as configurações, mas também o conteúdo de chaves e certificados para o cliente. Essas chaves e certificados estão originalmente em arquivos separados e portanto precisam ser copiados para o arquivo `.ovpn`. Dado o seguinte trecho de um modelo `.ovpn`:

```
client
dev tun
remote 192.168.1.155 1194
<ca>
ca.crt
</ca>
<cert>
client.crt
</cert>
```

```
<key>
client.key
</key>
<tls-auth>
ta.key
</tls-auth>
```

Supondo-se que os arquivos `ca.crt`, `client.crt`, `client.key` e `ta.key` estejam no diretório atual, como a configuração do modelo seria modificada pelo `sed` de forma a substituir cada nome de arquivo pelo seu conteúdo?

O comando

```
sed -r -e 's/(^[^.]*)\.(crt|key)$/cat \1.\2/e' < client.template > client.ovpn
```

substitui qualquer linha que termine em `.crt` ou `.key` pelo conteúdo de um arquivo cujo nome seja igual ao da linha. A opção `-r` diz ao `sed` para usar expressões regulares estendidas, ao passo que o `e` no final da expressão diz ao `sed` para substituir as correspondências pela saída do comando `cat \1.\2`. Os agrupamentos `\1` e `\2` correspondem ao nome de arquivo e extensão encontrados na correspondência.