Sed por exemplos - Parte 2

Como tirar mais vantagens do editor de textos UNIX

Daniel Robbins
President and CEO, Gentoo Technologies, Inc.
Outubro de 2000
http://www-106.ibm.com/developerworks/linux/library/l-sed2.html

O sed é um editor de texto stream muito poderoso e compacto. Neste artigo, o segundo da série, Daniel mostra como usar o sed para efetuar substituições de strings, criar scripts sed maiores, e usar os comandos para anexar, inserir, e alterar linhas.

Conteúdo

O sed é um editor de stream UNIX muito útil (mas frequentemente esquecido). É ideal para editar arquivos em lote ou para criar shell scripts que modificam arquivos de formas poderosas. Este artigo continua o artigo anterior.

Substituição!

Vamos olhar um dos comandos mais úteis do sed, o comando de substituição. Usando este comando, podemos trocar uma string em particular ou o que combinar com uma expressão regular por outra string. Veja um exemplo do uso mais comum deste comando:

```
$ sed -e 's/foo/bar/' myfile.txt
```

O comando acima irá escrever o conteúdo de myfile.txt para stdout, com a primeira ocorrência de 'foo' (se houver alguma) de cada linha trocada pela string 'bar'. Note que eu disse *a primeira ocorrência em cada linha*, apesar de que normalmente é isto o que você quer. Normalmente, quando eu faço uma troca de string, eu quero que a mesma seja feita globalmente. Ou seja, eu quero trocar *todas* as ocorrências em todas as linhas, conforme segue:

```
$ sed -e 's/foo/bar/g' myfile.txt
```

A opção 'g' adicional, após a última barra, informa ao sed que o mesmo deve executar uma substituição global.

Outras coisas que você precisa saber sobre o comando de substituição 's///'. Primeiro, ele é um comando, e somente um comando -- não há endereços especificados em nenhum dos endereços acima. Isto significa que o comando 's///' pode também ser usado com endereços para controlar a quais linhas ele será aplicado, como no exemplo abaixo:

```
$ sed -e '1,10s/enchantment/entrapment/g' myfile2.txt
```

O exemplo acima fará com que todas as ocorrências da palavra 'enchantment' sejam substituídas pela palavra 'entrapment', mas somente nas linhas um a dez, inclusive.

```
$ sed -e '/^$/,/^END/s/hills/mountais/g' myfile3.txt
```

Este exemplo irá trocar 'hills' por 'montains', mas somente em blocos de texto que começam com uma linha em branco, e terminando com uma linha que começa com os três caracters 'END', inclusive.

Outra coisa legal sobre o comando 's///' é que temos muitas opções quando se trata dos separadores '/'. se estamos fazendo uma substituição de strings e a expressão regular ou a string substituta possui várias barras, podemos trocar o separador especificando um caracter diferente após o 's'. Por exemplo, o seguinte comando irá trocar todas as ocorrências de /usr/local por /usr:

```
$ sed -e 's:/usr/local:/usr:g' mylist.txt
```

Neste exemplo, usamos o dois-pontos como separador. Se você precisar especificar o caracter separador na expressão regular, coloque uma contra-barra antes do mesmo.

Snafus no regexp

Até agora, somente executamos substituições de string simples. Apesar de ser útil, podemos também combinar expressões regulares. Por exemplo, o comando sed a seguir irá combinar qualquer frase que comece com '<' e termine com '>', e contenha qualquer número de caracteres no meio. Esta frase será apagada (substituída por uma string vazia):

```
$ sed -e 's/<.*>//g' myfile.html
```

Esta é uma boa tentativa de um script sed que irá remover todas as tags HTML de um arquivo, mas não irá funcionar bem, devido a um problema com as expressões regulares. A razão? Quando o sed tenta combinar a expressão regular em uma linha, ele encontra a *maior* combinação na linha. Isto não era um problema em meu artigo anterior, por que nós estávamos usando os comandos 'd' e 'p', que irão apagar ou escrever a linha inteira. Mas quando usamos o comando 's///', isto definitivamente faz uma grande diferença, por que todo o trecho que a expressão regular combinar será substituído pela string alvo, ou, neste caso, apagada. Isto significa que o exemplo acima irá transformar a seguinte linha:

```
<b>This</b> is what <b>I</b> meant.
```

nisto:

```
meant.
```

ao invés disto, que era o que queríamos fazer:

```
This is what I meant.
```

Felizmente, existe uma forma simples de corrigir isto. Em vez de escrever uma expressão regular que diga "um caracter '<' seguido por qualquer número de caracteres, e terminando com um caracter '>'", precisamos escrever uma regexp que diga "um caracter '<' seguido por qualquer número de caracteres não-'>', e terminando com um caracter '>'". Esta regexp terá o efeito de encontrar a menor combinação possível, ao invés da maior possível. O novo comando se parece com isto:

```
$ sed -e 's/<[^>]*>//g' myfile.html
```

No exemplo acima, o '[^>]' especifica um caracter "não-'>'", e o '*' que o segue completa a expressão, dando o significado "zero ou mais caracteres não-'>'". Teste este comando em alguns poucos arquivos HTML, faça o pipe deles para o comando more, e revise os resultados.

Mais combinação de caracteres

A sintaxe da expressão regular '[ ]' possui mais algumas opções adicionais. Para especificar um intervalo de caracters, pode-se utilizar um '-', desde que não esteja na primeira ou na última posição, como segue:

```
'[a-x]*'
```

Esta expressão regular irá combinar zero ou mais caracteres, desde que todos sejam 'a', 'b', 'c', ... 'v', 'w', 'x'. Além disso a classe de caracter '[:space:] está disponível para combinar os brancos. Segue uma lista compreensiva das classes de caracteres disponíveis:

Classe de Caracter	**Descrição**
[:alnum:]	Alfanumérica [a-z A-Z 0-9]
[:alpha:]	Alfabética [a-z A-Z]
[:blank:]	Espaços ou tabulações
[:cntrl:]	Qualquer caracter de controle
[:digit:]	Dígitos numéricos [0-9]
[:graph:]	Qualquer caracter visível (nenhum branco)
[:lower:]	Minúsculas [a-z]
[:print:]	Caracteres que não sejam de controle
[:punct:]	Caracteres de pontuação
[:space:]	Espaço em branco
[:upper:]	Maiúsculas [A-Z]
[:xdigit:]	Dígitos hexadecimais [0-9 a-z A-Z]

Usar classes de caracteres onde for possível traz vantagens, por que eles se adaptam melhor a 'locales' não-English (incluindo caracteres acentuados quando necessário, etc.).

Substituições avançadas

Olhamos até agora sobre como executar substituções diretas simples e mesmo razoavelmente complexas, mas o sed pode fazer mais. Podemos nos referir a partes ou à expressão regular que combinou inteira, e usar estas partes para construir a string de substituição. Como exemplo, digamos que você está respondendo a uma mensgem. O seguinte exmplo irá prefixar cada linha com "ralf said:":

```
sed -e 's/.*/raplh said: &/' origmsg.txt
```

A saída se parecerá com isto:

```
ralph said: Hiya Jim,
ralph said:
ralph said: I sure like this sed stuff!
falph said:
```

Neste exemplo, usamos o caracter '&' na string de substituição, que diz ao sed para inserir toda a expressão que combinou. Assim, o que quer que seja combinado por '.*' (o maior grupo de zero ou mais caracteres na linha, ou a linha inteira) pode ser inserido em qualquer lugar na string substituta, mesmo múltiplas linhas. Isto é excelente, mas o sed é ainda mais poderoso.

Contra-barra parêntesis

Ainda melhor que o '&', o comando 's///' nos permite definir *regiões* em nossa expressão regular, e podemos nos referir a estas regiões específicas em nossas strings substitutas. Por exemplo, digams que tenhamos um arquivo que contém o seguinte texto:

```
foo bar oni
eeny meeny miny
```

```
larry curly moe
jimmy the weasel
```

Agora, digamos que queiramos escrever um script que troque "eeny meeny miny" por "Victor eeny-meeny Von Miny", etc. Para isto, primeiro escrevemos uma expressão regular que irá combinar com as três strings, separadas por espaços:

```
'.* .* .*'
```

Assim. Agora, iremos definir regiões inserindo parêntesis escapados por contra-barras em torno de cada região de interesse:

```
'\(.*\) \(.*\) \(.*\)'
```

Esta expressão regular irá trabalhar da mesma forma que a anterior, exceto que ela irá definir três regiões lógicas às quais podemos nos referir em nossas strings substitutas. Aqui está a versão final do script:

```
$ sed -e 's/\(.*\) \(.*\) \(.*\)/Victor \1-\2 Von \3/' myfile.txt
```

Como você pode ver, nos referimos a cada região delimitada por parêntesis usando um '\x', onde x é o número da região, começando com um. A saída será como segue:

```
Victor foo-bar Von oni
Victor eeny-meeny Von miny
Victor larry-curly Von moe
Victor jimmy-the Von weasel
```

Conforme você se torna mais familiarizado com o sed, você poderá executar processamento de texto razoavelmente poderoso com um mínimo de esforço. Você pode querer pensar como você abordaria este problema usando sua linguagem de script favorita -- você conseguiria facilmente colocar a solução em uma linha?

Misturando as coisas

Conforme começamos a criar scripts sed mais complexos, precisamos a habilidade entrar mais de um comando. Existem várias formas de fazer isto. Primeiro, podemos usar ponto-e-vírgula entre os comandos. Por exemplo, esta série de comandos usa o comando '=', que faz o sed escrever o número da linha, bem como o comando 'p', que informa explicitamente ao sed para escrever a linha (já que estamos no modo '-n'):

```
$ sed -n -e '=;p' myfile.txt
```

Sempre que dois ou mais comandos são especificados, cada comando é aplicado (em ordem) a cada linha no arquivo. No exemplo acima, primeiro o comando '=' é aplicado à linha 1, e então o comando 'p' é aplicado. Então o sed avança para a linha 2, e repete o processo. Apesar do ponto-e-vírgula ser útil, existem casos em que ele não funciona. Outra alternativa é usar duas opções -e para especificar dois comandos separados:

```
$ sed -n -e '=' -e 'p' myfile.txt
```

Entretanto, a medida que chegamos a comandos de apensar e inserir mais complexos, mesmo múltiplas opções '-e' não são de ajuda. Para scripts multilinhas complexos, a melhor forma é colocar seus comandos em um arquivo separado. A seguir referencie este script com a opção -f:

```
$ sed -n -f mycommands.sed myfile.txt
```

Este método, apesar de menos conveniente, sempre funciona.

Múltiplos comandos para um endereço

Às vezes você pode querer especificar múltiplos comandos que serão aplicados a um único endereçamento. Geralmente isto acontece quando você está executando muitos 's///' para transformar palavras ou sintaxe em

arquivos fonte. Para executar múltiplos comandos por endereço, entre os comandos sed em um arquivo, e use os caracteres '{ }' para agrupar os comandos, conforme segue:

```
1,20{
        s/[Ll]inux/GNU\/Linux/g
        s/samba/Samba/g
        s/posix/POSIX/g
}
```

O exemplo acima aplica três comandos de substituição nas linhas 1 à 20, inclusive. Você pode usar expressões regulares no endereçamento, ou uma combinação dos dois:

```
1,/^END/{
        s/[Ll]inux/GNU\/Linux/g
        s/samba/Samba/g
        s/posix/POSIX/g
        p
}
```

Este exemplo irá aplicar todos os comandos que estão entre '{ }' nas linhas que iniciam com a linha 1 até uma linha que comece com as letras "END", ou o fim do arquivo se "END" não for encontrado no arquivo fonte.

Apensar, inserir, e mudar linhas

Agora que estamos escrevendo scripts sed em arquivos separados, podemos tomar vantagem dos comandos que apensam, inserem, e alteram linhas. Estes comandos irão inserir uma linha após a linha atual, inserir uma linha antes da linha atual, ou substituir a linha atual no espaço de padrões. Eles também podem ser usados para inserir múltiplas linhas na saída. O comando que insere linhas é usado conforme segue:

```
i\
This line will be inserted before each line
```

Se você não especificar um endereço para este comando, ele será aplicado a cada linha e irá produzir uma saída assim:

```
This line will be inserted before each line
line 1 here
This line will be inserted before each line
line 2 here
This line will be inserted before each line
line 3 here
This line will be inserted before each line
line 4 here
```

Se você quer inserir múltiplas linhas antes da linha atual, pode-se adicionar linhas acrescentando uma contra-barra no fim da linha anterior, como em:

```
i\
insert this line\
and this one\
and this one\
and, uh, this one too.
```

O comando append funciona de forma similar, mas insere uma linha ou linhas após a linha atual no espaço de padrão. É usada conforme segue:

```
a\
insert this line after each line. Thanks! :)
```

Por outro lado, o comando "change line" irá *trocar* a linha atual no espaço de padrões, e é usado conforme segue:

```
c\
You're history, original line! Muhahaha!
```

Como os comandos append, insert, e change precisam ser entrados em múltiplas linhas, você vai querer colocar as mesmas em scripts sed e informar o sed a ler as mesmas usando a opção '-f'. Usando os outros métodos para passar comandos ao sed irão resultar em problemas.

Recursos

Sobre o sed:

- Leia os outros artigos do Daniel sobre o Sed: Sed por exemplos, parte 1 e parte 3
- Cheque o excelente sed FAQ, de Eric Pement
- Os fontes do sed 3.02 podem ser encontrados em ftp://ftp.gnu.org/pub/gnu/sed
- O novo sed 3.02.80 pode ser encontrado em alpha.gnu.org
- Eric Pement também tem uma utilíssima lista de one-liners sed, que qualquer aspirante a guru definitivamente deve dar uma olhada
- Se você prefere um livro, sed & awk, 2nd Edition é uma excelente escolha
- Talvez você queira ler a 7a. edição da página man do sed do UNIX (de cerca de 1978!)
- Estude o tutorial do sed de Felix von Leitner
- Leia o artigo "Text processing in Python", no *developerWorks*

Sobre expressões regulares:

- Estude o tutorial exclusivo do dW, o using regular expressions, para encontrar e modificar padrões de texto
- Veja o how-to sobre expressões regulares em Python.org.
- Veja também o overview of regular expressions da University of Kentucky.

Sobre o autor

Residindo em Albuquerque, Novo México, Daniel Robbins é o Chief Architect do Gentoo Project, CEO da Gentoo Technologies, Inc., o criador do Gentoo Linux, um Linux avançado para o PC, e o sistema Portage, um sistema de ports para o Linux de última geração. Ele também é um autor-contribuinte dos livros da Macmillan *Caldera OpenLinux Unleashed*, *SuSE Linux Unleashed*, e *Samba Unleashed*. Daniel está envolvido com computadores de alguma forma desde o segundo grau, quando ele foi exposto pela primeira vez à linguagem de programação Logo, bem como a uma dose podencialmente perigosa de Pac Man. Isto provavelmente explica por que ele tem servido desde então como Lead Graphic Artist na SONY Eletronic Publishing/ Psygnosis. Daniel gosta de passar o tempo com sua esposa, Mary, e sua nova filhinha, Hadassah. Ele pode ser encontrado no email drobbins@gentoo.org.