Sed por exemplos - Parte 3

Levando para o próximo nível: Tratamento de dados, estilo sed

Daniel Robbins
President and CEO, Gentoo Technologies, Inc.
Novembro de 2000
http://www-106.ibm.com/developerworks/linux/library/l-sed3.html

Em sua conclusão da série sobre o sed, Daniel Robbins dá um gostinho do que é o verdadeiro poder do sed. Após introduzir um punhado de scripts sed essenciais, ele irá demonstrar alguns scripts sed radicais, ao converder um arquivo .QIF do Quicken em um formato texto legível. Este script de conversão não é apenas funcional, ele também serve como um excelente exemplo do poder dos scripts sed.

Conteúdo

Sed Muscular

Em meu segundo artigo sobre o sed, eu apresentei alguns exemplos que demonstravam como o sed funcionava, mas bem poucos destes exemplos realmente faziam algo realmente *útil*. Neste artigo final sobre o sed, é hora de mudar este comportamento e colocar o sed em tarefas úteis. Eu irei mostrar vários exemplos excelentes que não só demonstram o poder do sed, mas também fazem algo legal (e útil). Por exemplo, na segunda metade do artigo, eu irei mostrar como eu projetei um script sed que converte um arquivo .QIF do programa financeiro Quicken, da Intuit, em um arquivo texto bem formatado. Antes disto, daremos uma olhada em alguns scripts menos complicados mas ainda assim úteis.

Tradução de textos

Nosso primeiro script prático converte texto no estilo Unix no formato do DOS/Windows. Como você provavelmente já sabe, os arquivos textos no DOS/Windows possuem um CR (retorno de carro) e um LF (alimentação de linha) no fim de cada linha, enquanto que o UNIX tem somente um LF. Acontecerão várias oportunidades em que você precisará mover algum texto UNIX para um sistema Windows, e este script irá fazer a conversão de formato necessária para você.

```
$ sed -e 's/$/\r/' myunix.txt > mydos.txt
```

Neste script, a expressão regular '$' combina com o fim da linha, e o '\r' informa ao sed que este deve inserir um retorno de carro antes deste. Insira um retorno de carro antes de uma alimentação de linha, e presto, um CR/LF termina cada linha. Note que o '\r' será substituído por um CR somente quando estiver usando o GNU sed 3.02.80

ou posterior. Se você ainda não instalou o GNU sed 3.02.80 ainda, leia meu primeiro artigo sobre o sed para ver algumas instruções sobre como fazer isto.

Eu não sei dizer quantas vezes eu baixei algum script de exemplo ou código C, só para descobrir que estava no formato DOS/Windows. Apesar de muitos programas não se importarem com arquivos textos no formato DOS/Windows, com CR/LF, vários programas definitivamente se importam -- o mais notável é o bash, que rejeita a entrada assim que encontra um retorno de carro. O seguinte script sed irá converter textos no formato DOS/Windows para o formato texto confiável do UNIX:

```
$ sed -e 's/.$//' mydos.txt > myunix.txt
```

A forma de trabalho deste script é simples: nossa expressão regular de substituição combina com o último caracter da linha, que é um retorno de carro. Trocamos ele por nada, fazendo com que seja totalmente apagado da saída. Se você usar este script e notr que o último caracter de todas as linhas foi apagado, é por que especificou um arquivo texto que já está no formato UNIX. Não precisa fazer a alteração!

Revertendo linhas

Aqui há outro script útil. Ele inverte as linhas de um arquivo, da mesma forma que o comando "tac" que é incluído na maioria das distribuições Linux. O nome "tac" pode ser meio errôneo, já que o "tac" não inverte a posição dos caracteres na linha (esquerda e direita), mas a posição das linhas no arquivo (para cima e para baixo). Fazer um "tac" no seguinte arquivo:

```
foo
bar
oni
```

irá produzir a seguinte saída:

```
oni
bar
foo
```

Podemos fazer o mesmo com o seguinte script sed:

```
$ sed -e '1!G;h;$!d' forward.txt > backward.txt
```

Você irá achar este script útil se estiver logado no FreeBSD, que não tem um comando "tac". É um script útil, e é uma boa idéia saber por que este script faz o que faz. Vamos dissecar o mesmo.

Explicando a reversão

Primeiro, este script contém três diferentes comandos sed, separados por ponto-e-vírgula: '1!G', 'h' e '$!d'. Primeiro, é hora de ter um bom entendimento dos endereços usados no primeiro e terceiro comandos. Se o primeiro comando fosse '1G', o comando 'G' seria aplicado somente na primeira linha. Entretanto, há um caracter '!' adicional -- este caracter '!' *nega* o endereço, significando que o comando 'G' será aplicado a todas as linhas *exceto* a primeira. No comando '$!d' temos uma situação similar. Se o comando fosse '$d', ele aplicaria o comando 'd' somente na última linha do arquivo (o endereço '$' é uma forma simples de especificar a última linha). Entretanto, com o '!', '$!d' aplicará o comando 'd' a *todas* as linhas, menos a última linha. Agora, tudo que precisamos e entender o que os comandos em si fazem.

Quando executamos nosso script de inversão de linhas no arquivo texto acima, o primeiro comando que é executado é 'h'. Este comando diz ao sed para copiar o conteúdo do espaço de padrões (o buffer que mantém a linha que está sendo trabalhada) para o espaço hold (um buffer temporário). Então o comando 'd' é executado, que deleta o "foo" do espaço de padrões, de forma que ele não é escrito depois que todos os comandos são executados para esta linha.

Agora, a linha dois. Após "bar" ter sido lido para o espaço de padrões, o comando 'G' é executado, que apensa o espaço hold ("foo\n") ao espaço de padrões ("bar\n"), resultando em "bar\nfoo\n" em nosso espaço de padrões. O

comando 'h' coloca de volta no espaço hold para segurança, e o 'd' apaga a linha do espaço de padrões de forma que a linha não é escrita.

Para a última linha "oni", os mesmos passos são repetidos, exceto que o conteúdo do espaço de padrões não é apagado (devido ao '$!' em frente ao 'd'), e o conteúdo do espaço de padrões (três linhas) é escrito em stdout.

Agora, é hora de fazer alguma conversão de dados poderosa com o sed.

Mágica sed no QIF

Nas últimas semanas, eu estive pensando em comprar uma cópia do Quicken para fazer o balanço da minha conta bancária. O Quicken é um programa financeiro bem legal, e certamente iria fazer o serviço otimamente. Mas, após pensar sobre isto, eu decidi que eu poderia facilmente escrever um programa que faria o balanço do meu talão de cheques. Mesmo por quê, eu pensei, eu sou um desenvolvedor de software!

Eu desenvolvi um programa de balanço de cheques bem legal (usando o awk) que calcula o balanço examinando um arquivo texto que contém todas minhas transações. Depois de um pouco de trabalho, eu melhorei ele de forma que eu podia controlar diferentes categorias de crédito e débito, exatamente da mesma forma que o Quicken faz. Mas havia mais uma funcionalidade que eu gostaria de acrescentar. Eu recentemente passei minhas contas para um banco que possui uma interface Web de controle de contas. Um dia, notei que o Web site do meu banco permitia que eu fizesse o download das informações da minha conta no formato QIF do Quicken. Em pouco tempo, eu decidi que seria realmente legal se eu pudesse converter esta informação para formato texto.

Uma estória de dois formatos

Antes de olhar o formato QIF, este é o formato do meu arquivo checkbook.txt:

```
28 Aug 2000     food    -       -       Y       Supermarket     30.94
25 Aug 2000     watr    -       103     Y       Check 103       52.86
```

Em meu arquivo, todos os campos são separados por uma ou mais tabulações, com uma transação por linha. Após a data, o próximo campo lista o tipo de despesa (ou "-" se é um item de entrada). O terceiro campo lista o tipo de entrada (ou "-" se é um item de despesa). A seguir, há um campo de número de cheque (novamente, um "-" se estiver vazio), um campo informando se a transação foi efetivada ("Y" ou "N"), um comentário e uma quantia de dólares. Agora, estamos prontos para dar uma olhada no formato AIF. Quando eu vi meu arquivo QIF em um visualizador de text, foi isto que eu vi:

```
!Type:Bank
D08/28/2000
T-8.15
N
PCHECKCARD SUPERMARKET
^
D08/28/2000
T-8.25
N
PCHECKCARD PUNJAB RESTAURANT
^
D08/28/2000
T-17.17
N
PCHECKCARD SUPERMARKET
```

Depois de examinar o arquivo, não era muito difícil de descobrir qual o formato -- ignorando a primeira linha, o formato é como segue:

```
D
T
N
```

```
P
^
 (this is the field separator)
```

Iniciando o processo

Quando você está trabalhando em um projeto sed significante como este, não se desencoraje -- o sed permite que você passe os dados gradualmente para sua forma final. Conforme você progride, pode refinar continuamente seu script sed até que sua saída está exatamente como você quer. Você não precisa fazer tudo certo na primeira tentativa.

Para inciar, eu criei um arquivo chamado "quiftrans.sed", e comecei a tratar os dados:

```
1d
/^^/d
s/[[:cntrl:]]//g
```

O primeiro comando, '1d', apaga a primeira linha, e o segundo comando remove aqueles caracters "^" da saída. A última linha remove qualquer caracter de controle que possa existir no arquivo. Já que estou tratando com um formato de arquivo estranho, quero eliminar o risco de encontrar qualquer caracter de controle. Até agora, tudo bem. Agora é hora de acrescentar algum processamento ao script básico:

```
1d
/^^/d
s/[[:cntrl:]]//g
/^D/ {
        s/^D\(.*\)/\1\tOUTY\tINNY\t/
        s/^01/Jan/
        s/^02/Feb/
        s/^03/Mar/
        s/^04/Apr/
        s/^05/May/
        s/^06/Jun/
        s/^07/Jul/
        s/^08/Aug/
        s/^09/Sep/
        s/^10/Oct/
        s/^11/Nov/
        s/^12/Dec/
        s:^\(.*\)/\(.*\)/\(.*\):\2 \1 \3:
}
```

Primeiro, eu acrescentei um endereço '/^D/' de forma que o sed irá começar a processar quando encontrar o primeiro caracter do campo data QIF, 'D'. Todos os comandos entre chaves serão executados em ordem assim que o sed ler uma linha assim no espaço de padrões.

A primeira linha entre chaves irá transformar uma linha:

```
D08/28/2000
```

em uma linha assim:

```
08/28/2000      OUTY    INNY
```

Obviamente, este formato ainda não está perfeito, mas está OK. Iremos refinar gradualmente o conteúdo do espaço de padrões conforme prosseguimos. As próximas 12 linhas tem o efeito de transofrmar a data em um formato de três letras, com a última linha removendo as três barras da data. Terminaremos com algo assim:

```
Aug 28 2000     OUTY    INNY
```

Os campos OUTY e INNY estão guardando espaço e serão substituídos posteriormente. Eu não posso especificá-los agora, por que se a quantia de dólares for negativa, terei de configurar OUTY e INNY como "misc" e "-", mas

se a quantia de dólares é positiva, irei trocá-los para "-" e "inco", respectivamente. Uma vez que a quantia de dólares ainda não foi lida, preciso usar algo para guardar o lugar para quando chegar a hora.

Refinando

Agora é hora de mais algum refinamento:

```
1d
/^^/d
s/[[:cntrl:]]//g
/^D/ {
        s/^D\(.*\)/\1\tOUTY\tINNY\t/
        s/^01/Jan/
        s/^02/Feb/
        s/^03/Mar/
        s/^04/Apr/
        s/^05/May/
        s/^06/Jun/
        s/^07/Jul/
        s/^08/Aug/
        s/^09/Sep/
        s/^10/Oct/
        s/^11/Nov/
        s/^12/Dec/
        s:^\(.*\)/\(.*\)/\(.*\):\2 \1 \3:
        N
        N
        N
        s/\nT\(.*\)\nN\(.*\)\nP\(.*\)/NUM\2NUM\t\tY\t\t\3\tAMT\1AMT/
        s/NUMNUM/-/
        s/NUM\([0-9]*\)NUM/\1/
        s/\([0-9]\),/\1/
}
```

As próximas sete linhas são um pouco complicadas, de forma que iremos cobrí-las em detalhe. Primeiro, temos três comandos 'N' em seqüência. O comando 'N' diz ao sed para ler *a próxima linha na entrada* e anexar a mesma ao espaço de padrões atual. Os três comandos 'N' fazem com que as próximas três linhas sejam anexadas ao bufer de espaço de padrões atual, e agora nossa linha se parece com algo assim:

```
28 Aug 2000     OUTY    INNY    \nT-8.15\nN\nPCHECKCARD SUPERMARKET
```

O espaço de padrões do sed ficou estranho -- precisamos remover o caracteres de nova linha extras e fazer alguma formatação extra. para isto, usamos o comando de substituição. O padrão que queremos combinar é este:

```
'\nT.*\nN.*\nP.*'
```

Este padrão irá combinar um caracter de nova linha, seguido por um 'T', seguido por zer ou mais caracteres, seguido por um caracter de nova linha, seguido por um 'N', seguido por qualquer número de caracteres e um caracter de nova linha, seguido por um 'P', seguidopor qualquer número de caracteres. Esta rgexp irá combinar com o conteúdo completo das três linhas que recém anexamos no espaço de padrões. Mas queremos reformatar esta região, não substituir ela totalmente. A quantia de dólares, o número do cheque (se houver) e a descrição precisam reaparecer em nossa string de substituição. Para isto, cercamos estas "partes interessantes" com parêntesis escapados, asim podemos nos referir a elas em nossa string de substituição (usando '\1', '\2', e '\3' para informar ao sed onde inserir as mesmas). Aqui está o comando final:

```
s/\nT\(.*\)\nN\(.*\)\nP\(.*\)/NUM\2NUM\t\tY\t\t\3\tAMT\1AMT/
```

Este comando transforma nossa linha em:

```
28 Aug 2000     OUTY    INNY    NUMNUM  Y       CHECKCARD SUPERMARKET   AMT-8.15AMT
```

apesar desta linha estar bem melhor, existem algumas coisas que parecem meio ...interessantes, à primeira vista. A primeira coisa é aquela string boba, "NUMNUM" -- para quê ela serve? Inspecionando as próximas duas linhas do script sed, você descobre que iremos trocar "NUMNUM" por um "-", enquanto "NUM"<número>"NUM" será substituído por <número>. Como você pode ver, cercar o número do cheque com uma tag boba nos permite inserir convenientemente um "-" se o campo está vazio.

Toques finais

A última linha remove uma vírgula que segue um número. Isto irá converter quantias em dólar como "3,231.00" em "3231.00", que é o formato que eu uso. Agora, é hora de dar uma olhada no script final, o script em produção:

O script QIF para texto final

```
1d
/^^/d
s/[[:cntrl:]]//g
/^D/ {
        s/^D\(.*\)/\1\tOUTY\tINNY\t/
        s/^01/Jan/
        s/^02/Feb/
        s/^03/Mar/
        s/^04/Apr/
        s/^05/May/
        s/^06/Jun/
        s/^07/Jul/
        s/^08/Aug/
        s/^09/Sep/
        s/^10/Oct/
        s/^11/Nov/
        s/^12/Dec/
        s:^\(.*\)/\(.*\)/\(.*\):\2 \1 \3:
        N
        N
        N
        s/\nT\(.*\)\nN\(.*\)\nP\(.*\)/NUM\2NUM\t\tY\t\t\3\tAMT\1AMT/
        s/NUMNUM/-/
        s/NUM\([0-9]*\)NUM/\1/
        s/\([0-9]\),/\1/
        /AMT-[0-9]*.[0-9]*AMT/b fixnegs
        s/AMT\(.*\)AMT/\1/
        s/OUTY/-/
        s/INNY/inco/
        b done
:fixnegs
        s/AMT-\(.*\)AMT/\1/
        s/OUTY/misc/
        s/INNY/-/
:done
}
```

As onze linhas finais usam substituições e alguma funcionalidade de desvio para aperfeiçoar a saída. Vamos olhar primeiro nesta linha:

```
/AMT-[0-9]*.[0-9]*AMT/b fixnegs
```

Esta linha contém um comando de desvio, que está no formato "/regexp/b label". Se o espaço de padrões combina com a regexp, o sed irá desviar para o endereço fixnegs. Você deve poder encontrar este endereço facilmetne, que aparece como ":fixnets" no código. Se a regexp não combinar, o processamento continua normalmente com o próximo comando.

Agora que você entende o funcionamento do comando em si, vamos dar uma olhada nos desvios. Se você olhar a expressão regular do desvio, verá que ele combinará com a string 'AMT', seguida por um '-', seguida por qualquer

número de dígitos, um '.', qualquer número de dígitos e 'AMT'. Estou certo que você percebeu, esta regexp trata especificamente de uma quantia de dólares negativa. Um pouco antes nós havíamos cercado a quantia de dólares com strings 'AMT' para podermos encontrar a mesma facilmente, mais tarde. Como a regexp somente combina quantias de dólares que iniciam com um '-', nosso desvio só irá acontecer se estivermos tratando de um débito. Se estamos tratando um débito, OUTY deve ser passado para 'misc', INNY deve ser passado para '-', e o sinal negativo na frente da quantia de débito deve ser removida. Se você seguir o código, verá que é exatamente isto que acontece. Se o desvio não é executado, OUTY é substituído por um '-', e INNY é substituído por 'inco'. Terminamos! nossa linha de saída está agora perfeita:

```
 28 Aug 2000    misc    -       -       Y     CHECKCARD SUPERMARKET     -8.15
```

Não fique confuso

Como você pode ver, a conversão de dados usando o sed não é tão difícil, desde que você aborde o problema de uma forma incremental. Não tente fazer tudo com um único comando sed, ou tudo de uma só vez. Ao invés disto, trabalhe gradualmente em direção ao objetivo, e continue a melhorar seu script sed até que sua saída se pareça exatamente com o que você queria. O sed tem muito poder, e eu espero que você familiarize-se com suas funcionalidades, e você verá crescer o seu domínio do sed!

Recursos

Sobre o sed:

- Leia os outros artigos do Daniel sobre o Sed: Sed por exemplos, parte 1 e parte 2
- 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.