Introducción a la manipulación de texto en sistemas basados en UNIX

Utilizar herramientas estándar

Brad Yoes (wyoes@us.ibm.com)
Migration Engineer
IBM China

14-08-2012

Esta introducción a la manipulación de texto en plataformas UNIX provee una visión general de algunos comandos comunes ampliamente disponibles y del estándar instalado en la mayoría de los releases basados en UNIX. Muchas veces se ignoran estas herramientas estándar porque se prefieren los procesadores de texto más modernos como Perl, Python o Ruby, que no siempre están instalados en un sistema. Una reseña introductoria de estas herramientas ayuda a los profesionales que están aprendiendo UNIX o Linux, o a quienes estén en busca de actualizar sus conocimientos previos.

Introducción

Un postulado básico de la filosofía UNIX es crear programas (o procesos) que hacen una sola cosa y la hacen bien. Es una filosofía que exige pensar cuidadosamente acerca de las interfaces y las formas de unir estos procesos más pequeños (y con suerte, más simples) para crear resultados útiles. Normalmente, datos textuales fluyen entre estas interfaces. Con el tiempo se han desarrollado herramientas de procesamiento de texto cada vez más avanzadas. En cuanto a los lenguajes, al principio estaba perl, después llegó python y ruby. Mientras que éstos y otros lenguajes son procesadores de texto muy eficientes, dichas herramientas no siempre están disponibles, especialmente en el entorno de producción. En este artículo se hacen demostraciones de un grupo de comandos básicos de procesamiento de texto UNIX y se los utilizará individualmente o en conjunto entre sí para resolver problemas que también pueden ser abordados con lenguajes más recientes. Para muchos, ver un ejemplo brinda más información que leer explicaciones interminables. Por favor, observe que dada la variedad de sistemas UNIX y similares a UNIX que hay disponibles, las banderas de comandos, el comportamiento de programa y el resultado difieren entre las implementaciones.

Uso de cat

El comando `cat` es uno de los comandos más básicos. Se usa para crear, adjuntar, mostrar y concatenar archivos.

Podemos crear un archivo con cat utilizando '>' para redirigir una entrada estándar (stdin) a un archivo. Utilizar el operador '>' trunca los contenidos del archivo de salida especificado. El texto ingresado después de eso es redireccionado al archivo especificado a la derecha del operador '>'. Control-d señala un fin-de-archivo, devolviendo el control a shell.

Ejemplo de cat para crear un archivo:

```
$ cat > grocery.list
apples
bananas
plums
<ctrl-d>
$
```

Use el operador '>>' para adjuntar entradas estándar a un archivo existente.

Ejemplo de cat para adjuntar un archivo:

```
$ cat >> grocery.list
carrots
<ctrl-d>
```

Examine los contenidos del archivo grocery.list utilizando cat sin banderas. Note cómo los contenidos del archivo incluyen entradas de la redirección y adjuntan ejemplos de operador.

Ejemplo de cat sin banderas:

```
$ cat grocery.list
apples
bananas
plums
carrots
```

El comando cat se puede utilizar para enumerar las líneas de un archivo.

Ejemplo de cat para contar líneas:

```
$ cat -n grocery.list
    1  apples
    2  bananas
    3  plums
    4  carrots
```

Uso de nl

El filtro nl lee líneas de stdin o de los archivos especificados. El resultado se escribe en stdout, y puede ser redireccionado a un archivo o a otro proceso. El comportamiento de nl se controla a través de varias opciones de línea de comando.

De manera predeterminada, nl cuenta las líneas que son similares a cat -n.

Ejemplo de uso predeterminado de nl:

```
$nl grocery.list
    1  apples
    2  bananas
    3  plums
    4  carrots
```

Use la bandera -b para especificar las líneas a numerar. Esta bandera toma un “tipo” como argumento. El tipo le indica a `nl` qué líneas necesitan ser numeradas – use ‘a’ para numerar todas las líneas, ‘t’ le dice a `nl` que no numere las líneas vacías o las líneas que sólo son espacio blanco, ‘n’ especifica que no hay que numerar líneas. En el ejemplo se muestra un tipo de ‘p’ de patrón. `nl` numera las líneas especificadas por un patrón de expresión regular, en este caso, las líneas comienzan con la letras ‘a’ o ‘b’.

Ejemplo de nl para numerar líneas conforme a una expresión regular:

```
$ nl -b p^[ba] grocery.list
    1  apples
    2  bananas
      plums
      carrots
```

De manera predeterminada, `nl` separa el número de la línea del texto usando una tabulación. Use -s para especificar un delimitador distinto, como por ejemplo el signo ‘=’.

Ejemplo de nl para especificar un delimitador:

```
$nl –s= grocery.list
    1=apples
    2=bananas
    3=plums
    4=carrots
```

Uso de wc

El comando `wc` (wordcount) cuenta el número de líneas, palabras (separadas por espacio blanco), caracteres en filas especificadas o de `stdin`.

Ejemplos de uso de wc:

```
$wc grocery.list
       4       4      29 grocery.list
$wc -l grocery.list
       4 grocery.list
$wc -w grocery.list
       4 grocery.list
$wc -c grocery.list
      29 grocery.list
```

Usando grep

El comando `grep` busca archivos especificados o `stdin` en busca de patrones que concuerden con expresiones dadas. La salida de `grep` es controlada por varias banderas de opción.

A modo de demostración, se ha creado un archivo nuevo para usar grocery.list.

```
$cat grocery.list2
Apple Sauce
wild rice
black beans
kidney beans
dry apples
```

Ejemplo del uso básico de grep:

```
$ grep apple grocery.list grocery.list2
grocery.list:apples
grocery.list2:dry apples
```

grep tiene un número considerable de banderas de opción. A continuación se presentan algunos ejemplos que demuestran el uso de ciertas opciones.

Para mostrar el nombre de archivo (cuando hay múltiples archivos) con número de líneas donde fue encontrado el patrón – en este caso, contar el número de líneas en que aparece la palabra ‘apple’ en cada archivo.

Ejemplo de grep - contar el número de concordancias en los archivos:

```
$ grep -c apple grocery.list grocery.list2
grocery.list:1
grocery.list2:1
```

Al buscar en múltiples archivos, utilizar la opción -h suprime la impresión de nombre de archivo como parte del resultado.

Ejemplo de grep - suprimir nombre de archivo en resultado:

```
$ grep -h apple grocery.list grocery.list2
apples
dry apples
```

En muchas situaciones se prefiere una búsqueda sin diferenciación de mayúsculas y minúsculas. El comando grep tiene la opción -i para ignorar la diferenciación de mayúsculas y minúsculas al hacer las búsquedas.

Ejemplo de grep – sin diferenciación de mayúsculas y minúsculas:

```
$ grep -i apple grocery.list grocery.list2

grocery.list:apples
grocery.list2:Apple Sauce
grocery.list2:dry apples
```

A veces sólo se necesita imprimir el nombre de archivo y no la línea de concordancia de patrón. grep provee la opción -l para imprimir sólo los nombres de archivo que contienen líneas con un patrón concordante.

Ejemplo de grep – sólo nombres de archivo:

```
$ grep -l carrot grocery.list grocery.list2
grocery.list
```

Los números de línea se pueden proveer como parte del resultado. Use la opción -n para incluir los números de línea.

Ejemplo de grep – incluir números de línea:

```
$ grep -n carrot grocery.list grocery.list2
grocery.list:4:carrots
```

Hay ocasiones en las que el resultado deseado son las líneas que no concuerdan con el patrón. Para esos casos, utilice la opción -v.

Ejemplo de grep – concordancia invertida:

```
$ grep -v beans grocery.list2
Apple Sauce
wild rice
dry apples
```

A veces el patrón deseado forma una “palabra” rodeada de espacio blanco u otros caracteres como el guión o el paréntesis. La mayoría de las versiones de `grep` provee una opción -w para facilitar la escritura de búsquedas para estos patrones.

Ejemplo de grep – concordancia de palabra:

```
$ grep -w apples grocery.list grocery.list2
grocery.list:apples
grocery.list2:dry apples
```

Corrientes, tuberías, redirecciones, tee, documento-aquí

En UNIX un terminal contiene tres corrientes de forma predeterminada, una para entrada y dos corrientes basadas en salida. La corriente de entrada se llama `stdin`, y generalmente se correlaciona con el teclado (se pueden usar otros dispositivos de entrada o se podría redirigir desde otro proceso). La corriente de salida se llama `stdout`, y generalmente se imprime en el terminal, o la salida es utilizada por otro proceso (como `stdin`). La otra corriente de salida `stderr` utilizada principalmente para la información de estado se imprime generalmente en el terminal como `stdout`. Como cada una de estas corrientes tiene su propio descriptor de archivo, cada uno puede ser redirigido o redireccionado separadamente del otro, incluso si todos están conectados al terminal. Los descriptores de archivo para cada una de estas corrientes son:

- stdin = 0
- stdout = 1
- stderr = 2

Estas corrientes pueden ser redirigidas o redireccionadas a archivos o a otros procesos. Comúnmente, este concepto se conoce como construir una tubería. Por ejemplo, un programador puede querer fusionar las corrientes `stdout` y `stderr` y luego mostrarlas en el terminal pero también guardar los resultados en un archivo para examinar problemas de build. Al usar `2>&1` la corriente `stderr` con descriptor de archivo `2` es redireccionada a `&1`, un ‘puntero’ de la corriente `stdout` . Esto fusiona eficazmente `stderr` en `stdout`. Usar el símbolo ‘|’ indica una conexión. Una tubería enlaza `stdout` desde el proceso de la izquierda (`make`) a `stdin` del proceso de la derecha (`tee`). El comando `tee` duplica la corriente (fusionada) `stdout` enviando los datos al terminal y a un archivo, en este ejemplo, llamado build.log.

Ejemplo de fusión y división de corrientes estándar:

```
$ make –f build_example.mk 2>&1 | tee build.log
```

En otro ejemplo de redireccionamiento, se hace una copia de un archivo de texto utilizando el comando cat y un poco de redireccionamiento de corriente.

Ejemplo de redireccionamiento para hacer un archivo de copia de seguridad:

```
$ cat < grocery.list > grocery.list.bak
```

Antes utilizamos el comando `nl` para agregar números de líneas a un archivo mostrado en `stdout`. La tubería se puede utilizar para enviar la corriente `stdout` (desde grocery.list de cat) a otro proceso, en este caso, el comando `nl` .

Ejemplo de tubería simple a nl:

```
$ cat grocery.list | nl
    1  apples
    2  bananas
    3  plums
    4  carrots
```

Otro ejemplo mostrado antes fue cómo hacer una búsqueda sin diferenciación de mayúsculas y minúsculas de un archivo para un patrón. Esto se puede hacer usando redireccionamiento - en este caso desde `stdin`, o usando una tubería, de manera similar al ejemplo anterior de tubería simple.

Ejemplo con grep - redireccionamiento stdin y conexión:

```
$ grep -i apple < grocery.list2
Apple Sauce
dry apples
$cat grocery.list2 | grep -i apple
Apple Sauce
dry apples
```

En algunas situaciones un bloque de texto será redireccionado a un comando o a un archivo como parte de un script. Un mecanismo para lograr esto es el uso de ‘documento-aquí’ o ‘here-doc’. Para incorporar documento-aquí en un script, se utiliza el operador ‘`<<`’ para redireccionar el texto siguiente hasta que alcanza el delimitador fin-de-archivo. El delimitador es especificado después del operador `<<` .

Ejemplo de documento-aquí básico en la línea de comando:

```
$ cat << EOF
> oranges
> mangos
> pinapples
> EOF
oranges
mangos
pinapples
```

Este resultado se puede redireccionar a un archivo, en este ejemplo, el delimitador cambió de ‘EOF’ a ‘!’. Luego, el comando `tr` (explicado después) se usa para cambiar a mayúsculas las letras con documento-aquí.

Ejemplo de documento-aquí básico redireccionado a un archivo:

```
cat << ! > grocery.list3
oranges
mangos
pinapples
!
$ cat grocery.list3
oranges
mangos
pinapples
$tr [:lower:] [:upper:] << !
> onions
> !
ONIONS
```

Usando head y tail

Los comandos head y tail se usan para examinar las partes superiores (cabecera) o inferiores (pie) de los archivos. Para mostrar la parte superior de dos líneas y la parte inferior de dos líneas de un archivo use la bandera de opción -n con estos comandos, respectivamente. De forma similar, la opción -c muestra los primeros o los últimos caracteres del archivo.

Ejemplo de uso básico de los comandos head y tail:

```
$ head -n2 grocery.list
apples
bananas
$ tail -n2 grocery.list
plums
carrots
$ head -c12 grocery.list
apples
banan
$ tail -c12 grocery.list
ums
carrots
```

Un uso común del comando tail es la observación de archivos de registro o la salida de procesos en ejecución para ver si hay problemas, o para notar cuándo termina un proceso. La opción -f (tail –f) hace que tail siga vigilando la corriente incluso después de alcanzar el marcador fin-de-archivo, y continúe mostrando resultados cuando la corriente no contiene más datos.

Usando tr

El comando tr se utiliza para traducir caracteres desde stdin, mostrándolos en stdout. En su forma general, tr toma dos conjuntos de caracteres y reemplaza caracteres del primer conjunto con caracteres del segundo conjunto. Un número de clases (conjuntos) de caracteres predefinidos están disponibles para ser usados por tr, y algunos otros comandos.

Estas clases son:

- alnum - caracteres alfanuméricos
- alpha - caracteres alfabéticos

- blank - caracteres de espacio blanco
- cntrl - caracteres de control
- digit - caracteres numéricos
- graph - caracteres gráficos
- lower - caracteres alfabéticos en minúsculas
- print - caracteres imprimibles
- punct - caracteres de puntuación
- space - caracteres de espacio
- upper - caracteres en mayúsculas
- xdigit - caracteres hexadecimales

El comando `tr` puede traducir caracteres en minúsculas de una cadena a mayúsculas.

Ejemplo tr - cambiar una cadena a mayúsculas:

```
$ echo "Who is the standard text editor?" |tr [:lower:] [:upper:]
WHO IS THE STANDARD TEXT EDITOR?
```

`tr` se puede usar para eliminar caracteres con nombre de una cadena.

Ejemplo tr - eliminar caracteres de una cadena:

```
$ echo 'ed, of course!' |tr -d aeiou
d, f crs!
```

Use `tr` para traducir caracteres con nombre en una cadena a espacio. Cuando se encuentran caracteres con nombres múltiples en una secuencia, se traducen a un espacio simple.

El comportamiento de la bandera de opción -s difiere entre sistemas.

Ejemplo tr - traducir caracteres a un espacio:

```
$ echo 'The ed utility is the standard text editor.' |tr -s astu ' '
The ed ili y i he nd rd ex edi or.
```

La bandera de opción -s se puede usar para suprimir espacio blanco extra de una cadena.

```
$ echo 'extra     spaces – 5' | tr -s [:blank:]
extra spaces - 5
$ echo 'extra  tabs – 2' | tr -s [:blank:]
extra   tabs – 2
```

Un problema común al transferir archivos entre sistemas basados en UNIX y Windows son los delimitadores de línea. En sistemas UNIX el delimitador es una línea nueva, mientras que los sistemas Windows utilizan dos caracteres, un retorno de carro seguido por una línea nueva. Usar `tr` con algo de redireccionamiento es una forma de corregir este problema de formato.

Ejemplo tr - quitar retorno de carro:

```
$ tr -d '\r' < dosfile.txt > unixfile.txt
```

Uso de colrm

Al usar `colrm` se pueden cortar columnas de texto desde una corriente. En el primer ejemplo, se usa `colrm` para cortar de la columna 4 hasta el final de la línea para cada línea de la tubería. Luego, el mismo archivo se envía a `colrm` para retirar las columnas 4 a 5.

Ejemplo de colrm para quitar columnas:

```
$ cat grocery.list |colrm 4
app
ban
plu
car
$ cat grocery.list |colrm 4 5
apps
banas
plu
carts
```

Uso de expand y unexpand

El comando `expand` cambia las tabulaciones a espacios, mientras que `unexpand` cambia espacios a tabulaciones. Estos comandos sacan el resultado de `stdin` o de los archivos nombrados en la línea de comando. Utilizando la opción -t se pueden establecer una o más posiciones de tabulador.

Ejemplo de expand y unexpand:

```
$ cat grocery.list|head -2|nl|nl
    1       1  apples
    2       2  bananas
$ cat grocery.list|head -2|nl|nl|expand -t 5
    1       1    apples
    2       2    bananas
$ cat grocery.list|head -2|nl|nl|expand -t 5,20
    1               1 apples
    2               2 bananas
$ cat grocery.list|head -2|nl|nl|expand -t 5,20|unexpand -t 1,5
            1               1 apples
            2               2 bananas
```

Uso de comm, cmp, y diff

Para demostrar estos comandos se han creado dos archivos nuevos.

Crear archivos para la demostración:

```
cat << EOF > dummy_file1.dat
011 IBM 174.99
012 INTC 22.69
013 SAP 59.37
014 VMW 102.92
EOF
cat << EOF > dummy_file2.dat
011  IBM 174.99
012 INTC 22.78
013 SAP 59.37
014 vmw 102.92
EOF
```

El comando diff compara dos archivos, informando las deferencias que hay entre ellos. diff toma un número de banderas de opciones. En el ejemplo siguiente, se muestra primero un diff predeterminado seguido por un diff usando la opción -w que ignora espacio blanco, luego termina con un ejemplo de la bandera de opción -i que ignora las diferencias de mayúsculas y minúsculas cuando hace las comparaciones.

Ejemplos del comando diff:

```
$ diff dummy_file1.dat dummy_file2.dat
1,2c1,2
< 011 IBM 174.99
< 012 INTC 22.69
---
> 011  IBM 174.99
> 012 INTC 22.78
4c4
< 014 VMW 102.92
---
> 014 vmw 102.92

$ diff -w dummy_file1.dat dummy_file2.dat
2c2
< 012 INTC 22.69
---
> 012 INTC 22.78
4c4
< 014 VMW 102.92
---
> 014 vmw 102.92

$ diff -i dummy_file1.dat dummy_file2.dat
1,2c1,2
< 011 IBM 174.99
< 012 INTC 22.69
---
> 011  IBM 174.99
> 012 INTC 22.78
```

El comando comm compara dos archivos pero se comporta de forma bastante distinta que diff. comm produce tres columnas de resultado - sólo líneas de file1 (columna 1), sólo líneas de file2 (columna 2) y líneas comunes a los dos archivos (columna 3). Las banderas de opción se pueden utilizar para suprimir columnas de salida. Probablemente, este comando es más útil para suprimir la columna 1 y la columna 2, mostrando únicamente las líneas que son comunes a ambos archivos, como se muestra debajo.

Ejemplo de comando comm:

```
$ comm dummy_file1.dat dummy_file2.dat
        011  IBM 174.99
011 IBM 174.99
012 INTC 22.69
        012 INTC 22.78
                013 SAP 59.37
014 VMW 102.92
        014 vmw 102.92

$ comm -12 dummy_file1.dat dummy_file2.dat
013 SAP 59.37
```

El comando cmp también compara dos archivos. Sin embargo, a diferencia de comm o diff, el comando cmp (de forma predeterminada) informa el primer número de byte y de línea donde los dos archivos difieren.

Ejemplo de comando cmp:

```
$ cmp dummy_file1.dat dummy_file2.dat
dummy_file1.dat dummy_file2.dat differ: char 5, line 1
```

Usar fold

Cuando se usa el comando `fold` las líneas se dividen en un ancho especificado. Originariamente, este comando se utilizaba para ayudar a dar formato a texto para dispositivos de salida con ancho fijo que no tenían la capacidad de ajustar texto. La bandera de opción -w permite la especificación del ancho de una línea para usar en lugar de las 80 columnas predeterminadas.

Ejemplo de uso de fold:

```
$ fold -w8 dummy_file1.dat
011 IBM
174.99
012 INTC
 22.69
013 SAP
59.37
014 VMW
102.92
```

Usando paste

El comando `paste` se utiliza para alinear archivos lado a lado, fusionando registros de cada archivo, respectivamente. Cuando se usa redireccionamiento se pueden crear nuevas filas uniendo cada registro de un archivo con registros en otro archivos.

Crear archivos para la demostración:

```
cat << EOF > dummy1.txt
IBM
INTC
SAP
VMW
EOF
cat << EOF > dummy2.txt
174.99
22.69
59.37
102.92
EOF
```

Ejemplo 1 de paste - líneas de múltiples archivos:

```
$ paste dummy1.txt dummy2.txt grocery.list
IBM     174.99  apples
INTC    22.69   bananas
SAP     59.37   plums
VMW     102.92  carrots
```

Hay una bandera de opción -s para procesar los archivos de a uno (en serie) en lugar de hacerlo en paralelo. Observe debajo cómo las columnas se alinean con las filas del ejemplo anterior.

Ejemplo 2 de paste - líneas de múltiples archivos:

```
$ paste -s dummy1.txt dummy2.txt grocery.list
IBM     INTC    SAP     VMW
174.99  22.69   59.37   102.92
apples  bananas plums   carrots
```

Si se especifica sólo un archivo, o si `paste` está procesando `stdin`, de forma predeterminada, la entrada es presentada en una columna. Cuando se usa la bandera de opción -s, el resultado se presenta en una línea. Dado que el resultado está condensado en una línea sola, use un delimitador para separar los campos retornados (tabulación es el delimitador predeterminado). En este ejemplo, el comando `find` se usa para localizar directorios donde es más probable encontrar las bibliotecas de 64 bits, y desarrolla una ruta adecuada para adjuntar a una variable $LD_LIBRARY_PATH.

Ejemplos de paste - con delimitador:

```
$ find /usr -name lib64 -type d|paste -s -d:
/usr/lib/qt3/lib64:/usr/lib/debug/usr/lib64:/usr/X11R6/lib/X11/locale/lib64:/usr/X11R6/
lib64:/usr/lib64:/usr/local/ibm/gsk7_64/lib64:/usr/local/lib64

$ paste -d, dummy1.txt dummy2.txt
IBM,174.99
INTC,22.69
SAP,59.37
VMW,102.92
```

Usando bc

Para usar una forma sencilla de hacer cálculos en shell, considere `bc`, la “calculadora básica” o “calculadora de escritorio”. Algunos shells ofrecen formas de hacer cálculos nativamente, otros dependen de `expr` para evaluar las expresiones. Cuando usa `bc`, los cálculos se pueden transportar entre shells y sistemas UNIX, pero tenga cuidado con las extensiones de proveedores.

Ejemplo de bc – cálculos simples:

```
$ echo 2+3|bc
5

$ echo 3*3+2|bc
11

$ VAR1=$(echo 2^8|bc)
$ echo $VAR1
256

$ echo "(1+1)^8"|bc
256
```

`bc` puede realizar más que estos simples cálculos. Es un intérprete que define sus propias funciones, sintaxis y flujo de control internos y definidos por usuarios, de manera similar a un lenguaje de programación. De manera predeterminada, `bc` no incluye los dígitos a la derecha de los decimales. Aumente la precisión del resultado utilizando la variable especial `scale` . Como muestra el ejemplo, `bc` escala a números grandes e instrumenta una precisión prolongada. Use `obase` o `ibase` para controlar la base de conversión para los números de entrada y salida. En el ejemplo siguiente:

- `obase` cambia la base de salida predeterminada (diez), convirtiendo el resultado a hexadecimal

- para esta aproximación de raíz cuadrada de 2, scale especifica el número de dígitos a la derecha del decimal
- el soporte de números grandes se ilustra calculando 2 a la 128
- la función interna sqrt() es necesaria para calcular la raíz cuadrada de 2
- Desde ksh, calcule e imprima un porcentaje

Ejemplo de bc – más cálculos:

```
$ echo "obase=16; 2^8-1"|bc
FF

$ echo "99/70"|bc
1

$ echo "scale=20; 99/70"|bc
1.41428571428571428571

$ echo "scale=20;sqrt(2)"|bc
1.41421356237309504880

$ echo 2^128|bc
340282366920938463463374607431768211456

$ printf "Percentage: %2.2f%%\n" $(echo .9963*100|bc)
Percentage: 99.63%
```

La página principal de bc es detallada e incluye ejemplos.

Usando split

Una tarea útil del comando split es dividir grandes archivos de datos en archivos más pequeños para procesar. En este ejemplo se muestra que BigFile.dat tiene 165782 líneas que usan el comando wc . La bandera de la opción -l le indica a split el número máximo de líneas de cada archivo de salida. split permite especificar un prefijo para nombres de archivo de salida, debajo, BigFile_ es el prefijo especificado. Otras opciones permiten el control del sufijo y en BSD, una bandera de opción -p permite que se produzcan divisiones a una expresión regular como el comando csplit (división de contexto). Consulte la página principal para obtener más información.

Ejemplo de split:

```
$ wc BigFile.dat
165782  973580 42557440 BigFile.dat

$ split -l 15000 BigFile.dat BigFile_

$ wc BigFile*
  165782  973580 42557440 BigFile.dat
   15000   87835 3816746 BigFile_aa
   15000   88483 3837494 BigFile_ab
   15000   89071 3871589 BigFile_ac
   15000   88563 3877480 BigFile_ad
   15000   88229 3855486 BigFile_ae
    7514   43817 1908914 BigFile_af
  248296 1459578 63725149 total
```

Usando cut

El comando cut se utiliza para ‘recortar’ secciones basadas en columnas de un archivo o de datos redirigidos al mismo desde stdin. Corta datos en bytes (-b), caracteres (-c), o campos (-f) según lo especifique la lista.

Las listas de campos o de posiciones de byte/carácter son especificadas usando listas separadas por comas y guiones. Simplemente especifique la posición o el campo deseado si sólo se necesita el resultado de uno. Se puede especificar un rango de campos usando un guión de modo que 1-3 imprima los campos (o posiciones) 1 a 3, -2 imprima desde el comienzo de la línea hasta el campo 2 (ó byte/carácter 2), y 3- especifique a `cut` que imprima el campo (ó posición) 3 hasta el final de la línea. Se pueden especificar múltiples campos usando una coma. Otras banderas que pueden ser útiles son -d para especificar un delimitador de campo, y -s, para suprimir líneas sin delimitadores.

Ejemplos de cut:

```
$ cat << EOF > dummy_cut.dat
# this is a data file
ID,Name,Score
13BA,John Smith,100
24BC,Mary Jones,95
34BR,Larry Jones,94
36FT,Joe Ruiz,93
40RM,Kay Smith,91
EOF

$ cat dummy_cut.dat |cut -d, -f1,3
# this is a data file
ID,Score
13BA,100
24BC,95
34BR,94
36FT,93
40RM,91

$ cat dummy_cut.dat |cut -b6-
s is a data file
me,Score
John Smith,100
Mary Jones,95
Larry Jones,94
Joe Ruiz,93
Kay Smith,91

$ cat dummy_cut.dat |cut -f1- -d, -s
ID,Name,Score
13BA,John Smith,100
24BC,Mary Jones,95
34BR,Larry Jones,94
36FT,Joe Ruiz,93
40RM,Kay Smith,91
```

Usando uniq

El comando `uniq` se usa típicamente para enumerar listas de forma única desde una fuente de entrada (generalmente un archivo o `stdin`). Para operar apropiadamente, las líneas duplicadas deben ser posicionadas contiguamente en la entrada. Normalmente, la entrada al comando `uniq` se ordena, por lo tanto las líneas duplicadas se alinean. Dos banderas comunes que se utilizan con el comando `uniq` son -c, para imprimir el conteo del número de veces que apareció cada línea, y -d se puede utilizar para mostrar una instancia de líneas duplicadas.

Ejemplos de uniq:

```
$ cat << EOF > dummy_uniq.dat

13BAR   Smith   John    100
```

```
13BAR   Smith   John    100
24BC    Jone    Mary    95
34BRR   Jones   Larry   94
36FT    Ruiz    Joe     93
40REM   Smith   Kay     91
13BAR   Smith   John    100
99BAR   Smith   John    100
13XIV   Smith   Cindy   91


EOF

$ cat dummy_uniq.dat | uniq

13BAR   Smith   John    100
24BC    Jone    Mary    95
34BRR   Jones   Larry   94
36FT    Ruiz    Joe     93
40REM   Smith   Kay     91
13BAR   Smith   John    100
99BAR   Smith   John    100
13XIV   Smith   Cindy   91

$ cat dummy_uniq.dat | sort |uniq

13BAR   Smith   John    100
13XIV   Smith   Cindy   91
24BC    Jone    Mary    95
34BRR   Jones   Larry   94
36FT    Ruiz    Joe     93
40REM   Smith   Kay     91
99BAR   Smith   John    100

$ cat dummy_uniq.dat | sort |uniq -d

13BAR   Smith   John    100

$ cat dummy_uniq.dat | sort |uniq -c
  3
  3 13BAR   Smith   John    100
  1 13XIV   Smith   Cindy   91
  1 24BC    Jone    Mary    95
  1 34BRR   Jones   Larry   94
  1 36FT    Ruiz    Joe     93
  1 40REM   Smith   Kay     91
  1 99BAR   Smith   John    100
```

Usando sort

Para ordenar filas en `stdin` o un archivo en un orden en particular tal como alfabético o numérico, se puede usar el comando `sort` . De manera determinada, el resultado de `sort` se escribe en `stdout`. Las variables del entorno tales como LC_ALL, LC_COLLATE, ó LANG pueden afectar el resultado de `sort` y otros comandos. Observe cómo el archivo de ejemplo muestra 2 registros duplicados separados – un doble de IBM y el otro doble es una línea vacía.

Ejemplo de sort – comportamiento predeterminado:

```
$ cat << EOF > dummy_sort1.dat

014 VMW, 102.92
013 INTC, 22.69
012 sap,  59.37
011 IBM, 174.99
011 IBM, 174.99

EOF

$ sort dummy_sort1.dat


011 IBM, 174.99
011 IBM, 174.99
012 sap,  59.37
013 INTC, 22.69
014 VMW, 102.92
```

sort tiene una bandera excelente para sustituir el comando uniq en muchas circunstancias. La bandera de opción -u ordena el archivo, quita las filas duplicadas de modo que se produzca un listado de filas únicas de resultado.

Ejemplo de sort – ordenar único:

```
$ sort -u dummy_sort1.dat

011 IBM, 174.99
012 sap,  59.37
013 INTC, 22.69
014 VMW, 102.92
```

A veces se necesita el ordenamiento revertido de la entrada. De manera predeterminada, sort organiza el orden de menor a mayor (numérico) y en orden alfabético para los datos de caracteres. Use la bandera de opción -r para revertir el orden del ordenamiento predeterminado.

Ejemplo de sort – ordenar orden revertido:

```
$ sort -ru dummy_sort1.dat
014 VMW, 102.92
013 INTC, 22.69
012 sap,  59.37
011 IBM, 174.99
```

Distintas situaciones requieren que un archivo sea ordenado en ciertos campos o “claves”. Afortunadamente sort tiene la bandera de opción -k que permite especificar una llave de ordenamiento por posición. Los campos son separados por espacio blanco de forma determinada.

Ejemplo de sort – ordenar con una clave:

```
$ sort -k2 -u dummy_sort1.dat

011 IBM, 174.99
013 INTC, 22.69
014 VMW, 102.92
012 sap,  59.37
```

Cuando la diferenciación de mayúsculas y minúsculas representa un problema, sort provee la bandera de opción -f, que ignora mayúsculas y minúsculas al hacer comparaciones. Cuando se combinan banderas múltiples como se muestra debajo, algunas versiones de UNIX necesitan que estas banderas estén especificadas en un orden distinto.

Ejemplo de sort – ordenar ignorando mayúsculas y minúsculas:

```
$ sort -k2 -f -u dummy_sort1.dat

011 IBM, 174.99
013 INTC, 22.69
012 sap,  59.37
014 VMW, 102.92
```

Hasta ahora, todos los ordenamientos han sido del tipo alfabético. Cuando necesite ordenar los datos en orden numérico, use la bandera de opción -n.

Ejemplo de sort – ordenamiento numérico:

```
$ sort -n -k3 -u dummy_sort1.dat

013 INTC, 22.69
012 sap,  59.37
014 VMW, 102.92
011 IBM, 174.99
```

Algunas entradas pueden usar caracteres que no sean espacio blanco para separar campos en una fila. Use la bandera de opción -t para especificar un delimitador no predeterminado, como puede ser una coma.

Ejemplo de sort – ordenar campo usando delimitador no predeterminado:

```
$ sort -k2 -t"," -un dummy_sort1.dat

013 INTC, 22.69
012 sap,  59.37
014 VMW, 102.92
011 IBM, 174.99
```

Usando join

Cualquier persona familiarizada en la escritura de búsquedas de base de datos reconoce la utilidad del comando join . Al igual que la mayoría de los comando UNIX, el resultado se muestra en stdout. Para “combinar” archivos, se comparan los campos especificados de dos archivos línea por línea. Si hay campos especificados, join busca concordancias en los campos desde el comienzo de cada línea. El separador de campos predeterminado es el espacio blanco (en algunos sistemas es simplemente un espacio o espacios adyacentes). Cuando se produce una concordancia de campos, se escribe una línea de resultados por cada par de líneas con los campos concordantes. Para obtener resultados legítimos, los dos archivos tienen que estar ordenados por campos para que puedan concordar. No todos los sistemas implementan join de la misma forma.

Este ejemplo utiliza -t para especificar un separador de campo y muestra cómo combinar dos archivos en el primer campo (predeterminado) delimitado por comas. Los operadores de base de datos deberían reconocerlo como una combinación interna que muestra sólo las filas concordantes.

Ejemplo de join – usando delimitador de campo no predeterminado:

```
cat << EOF > dummy_join1.dat
011,IBM,Palmisano
012,INTC,Otellini
013,SAP,Snabe
014,VMW,Maritz
015,ORCL,Ellison
017,RHT,Whitehurst
EOF

cat << EOF > dummy_join2.dat
011,174.99,14.6
012,22.69,10.4
013,59.37,26.4
014,102.92,106.1
016,27.77,31.2
EOF

cat << EOF > dummy_join3.dat
IBM,Armonk
INTC,Santa Clara
SAP,Walldorf
VMW,Palo Alto
ORCL,Redwood City
EMC,Hopkinton
EOF

$ join -t, dummy_join1.dat dummy_join2.dat
011,IBM,Palmisano,174.99,14.6
012,INTC,Otellini,22.69,10.4
013,SAP,Snabe,59.37,26.4
014,VMW,Maritz,102.92,106.1
```

Para especificar campos para “combinar” en cada archivo se puede usar la bandera de opción -j[1,2] x (ó simplemente, -1 x ó -2 x). La bandera de opción -j1 2 ó -1 2 especifica el segundo campo del archivo uno, el primer archivo presentado en el comando. Este ejemplo muestra cómo combinar los archivos basados en el campo 1 en el primer archivo y el campo 2 en el segundo archivo, y también una combinación interna para concordar únicamente con las filas.

Ejemplo de join – campos especificados:

```
$ join -t, -j1 1 -j2 2 dummy_join3.dat dummy_join1.dat
IBM,Armonk,011,Palmisano
INTC,Santa Clara,012,Otellini
SAP,Walldorf,013,Snabe
VMW,Palo Alto,014,Maritz
ORCL,Redwood City,015,Ellison
```

Continuando con la idea de hacer ejemplos relativos a bases de datos, las banderas pueden utilizarse para producir una combinación externa de tabla izquierda. Left outer join incluye todas las filas del primer archivo o tabla izquierda y las filas que concuerdan en el segundo archivo o tabla. Use -a para incluir todas las filas del archivo especificado.

Ejemplo de join – left outer join:

```
$ join -t, -a1 dummy_join1.dat dummy_join2.dat
011,IBM,Palmisano,174.99,14.6
012,INTC,Otellini,22.69,10.4
013,SAP,Snabe,59.37,26.4
014,VMW,Maritz,102.92,106.1
015,ORCL,Ellison
017,RHT,Whitehurst
```

Full outer joins incluye todas las filas de ambos archivos o tablas, sin importar si los campos concuerdan. Se puede realizar una combinación externa completa especificando ambos archivos con la bandera de opción -a.

Ejemplo de join – full outer join:

```
$ join -t, -a1 -a2 -j1 2 -j2 1 dummy_join1.dat dummy_join3.dat
IBM,011,Palmisano,Armonk
INTC,012,Otellini,Santa Clara
SAP,013,Snabe,Walldorf
VMW,014,Maritz,Palo Alto
ORCL,015,Ellison,Redwood City
EMC,Hopkinton
017,RHT,Whitehurst
```

Usando sed

El editor de flujos `sed` es una herramienta útil de análisis y manipulación de texto que sirve para hacer transformaciones en archivos o flujos de datos. Lee texto línea por línea, aplicando los comando especificados en la línea de texto. De forma predeterminada, el resultado va a los comandos `stdout`. El comando `sed` usa operaciones de rendimiento tales como borrar texto de un almacenamiento intermedio, adjuntar o insertar texto en un almacenamiento intermedio, escribir en un archivo, transformar texto basado en expresiones regulares y más.

Un ejemplo básico de sustitución `sed` muestra la bandera de opción -e utilizada para especificar la expresión o el script de edición. Se pueden especificar múltiples expresiones o ediciones para una sola ejecución `sed` . Note los componentes de la edición `sed` . La “s” al principio de la edición indica que esto es un comando de sustitución. Cuando se usa “/” como delimitador se indica primero el patrón “IBM” a reemplazar. Luego, aparece el patrón de reemplazo entre dos delimitadores “/”. Por último, “g” indica que hay que hacer el cambio globalmente en el almacenamiento intermedio de texto actual. La tercera demostración de este ejemplo ilustra una combinación de tres ediciones: reemplazar barras invertidas por barras comunes, espacios por guiones bajos, y quitar caracteres de dos puntos – note cómo los caracteres de barra invertida “\” tienen los caracteres de escape.

Ejemplo de sed – sustitución básica / ediciones múltiples:

```
$ echo "IBM 174.99" |sed –e 's/IBM/International Business Machines/g'
International Business Machines 174.99

$ echo "Oracle DB"|sed -e 's/Oracle/IBM/g' -e 's/DB/DB2/g'
IBM DB2

$ echo "C:\Program Files\PuTTY\putty.exe"| sed -e 's/\\/\//g' -e 's/ /_/g' -e 's/://g'
C/Program_Files/PuTTY/putty.exe
```

En el ejemplo siguiente se configura un archivo para demostrar otro recurso de sed. Además de la sustitución, el filtrado es otro uso frecuente de sed. El comando UNIX grep es un filtro empleado comúnmente; no es raro descubrir múltiples formas de manipular el texto en la línea de comando. Este ejemplo muestra cómo utilizar el comando borrar de sed para quitar líneas que comienzan con “#” ó espacio blanco y luego “#”. Se muestra un ejemplo de grep utilizando el mismo patrón a modo de referencia.

Ejemplo de sed - filtrado:

```
cat << EOF > dummy_sed.txt
# top of file
  # the next line here
# Last Name, Phone
Smith, 555-1212
Jones, 555-5555 # last number
EOF

$ sed '/^[[:space:]]*#/d' dummy_sed.txt
Smith, 555-1212
Jones, 555-5555 # last number

$ grep -v  ^[[:space:]]*# dummy_sed.txt
Smith, 555-1212
Jones, 555-5555 # last number
```

Para entender mejor el comportamiento de sed se muestran algunos patrones más. Se crea un archivo para que los patrones puedan actuar en cierto texto. El primer patrón sed muestra cómo eliminar los últimos 4 caracteres de la cadenas (nombres de archivo) presentadas en el archivo. Luego, el patrón elimina todos los caracteres a la derecha del punto “.” que indican una extensión de archivo. Debajo se muestra un patrón para eliminar líneas vacías. Un carácter especial, el símbolo et “&” permite que el patrón de búsqueda sea utilizado como parte del resultado. En este ejemplo, IBM es parte del patrón de entrada y está especificado como parte del resultado usando el símbolo et. El último patrón demostrado en esta serie muestra cómo se puede usar sed para eliminar retornos de carro de un archivo de texto transferido desde un sistema basado en Windows. Se ingresa “^M” en el script o en la línea de comando presionando primero Control-v y luego pulsando control-m. Por favor, note que las características del terminal pueden afectar el ingreso de la combinación control-v, control-m.

Ejemplo de sed – más patrones:

```
cat << EOF > filelist.txt
PuTTY.exe

sftp.exe
netstat.exe
servernames.list
EOF

$ sed 's/....$//' filelist.txt
PuTTY

sftp
netstat
servernames.

$ sed 's/\..*$//g' filelist.txt
PuTTY

sftp
```

```
netstat
servernames

$ sed '/^$/d' filelist.txt
PuTTY.exe
sftp.exe
netstat.exe
servernames.list

$ echo "IBM 174.99" |sed 's/IBM/&-International Business Machines/g'
IBM-International Business Machines 174.99

$ cat dosfile.txt | sed 's/^M//' > unixfile.txt
```

sed puede operar en rangos de direcciones especificadas. Los siguientes ejemplos muestran algunas formas en que se pueden controlar las direcciones con sed. La bandera de opción “-n” suprime el comportamiento predeterminado de sed para mostrar cada línea de entrada como parte del resultado. En el primer ejemplo, sed opera en las líneas 4 a 7 desde el archivo. Note cómo sólo se muestra la primera fila o tabla presentada del archivo (líneas 4 a 7). Luego sed muestra sólo la primera y la última línea del archivo. La mayoría de las versiones de sed permiten que los patrones especifiquen un rango de direcciones para aplicar el comando. Note que en el resultado sólo las comas, y no los comentarios, son eliminadas de la tabla.

Ejemplo de sed – rangos de direcciones:

```
cat << EOF > dummy_table.frag

<p>This, is a paragraph.</p>
<table border="1">
<tr>
<td>row 1, 1st cell</td>
<td>row 1, 2nd cell</td>
</tr>
<tr>
<td>row 2, 1st cell</td>
<td>row 2, 2nd cell</td>
</tr>
</table>
<!--This, is another comment. -->
EOF

$ sed -n 4,7p dummy_table.frag
<tr>
<td>row 1, 1st cell</td>
<td>row 1, 2nd cell</td>
</tr>

$ sed -n -e 1p -e \$p dummy_table.frag
<!--This, is a comment. -->
<!--This, is another comment. -->

$ sed '/^&lt;table/,/^&lt;\/table/s/,//g' dummy_table.frag
<!--This, is a comment. -->
<p>This, is a paragraph.</p>
<table border="1">
<tr>
<td>row 1 1st cell</td>
<td>row 1 2nd cell</td>
</tr>
<tr>
<td>row 2 1st cell</td>
<td>row 2 2nd cell</td>
</tr>
</table>
```

```
<!--This, is another comment. -->
```

Los patrones que están dentro de una expresión pueden ser agrupados y luego referenciados como parte del resultado. Esto puede resultar útil en una variedad de contextos tales como el intercambio de valores o las variables posicionales. Los paréntesis se utilizan para resaltar patrones en la expresión y se los debe separar con el carácter de escape barra invertida \(pattern-here\). El patrón se referencia en otras partes de la expresión utilizando \n donde n es el número del patrón en el orden de los patrones marcados. Descomponer esta expresión en componentes hace que sea más fácil reconocer cómo funciona:

Patrón	Comentarios
/^#.*$/d	eliminar de líneas de resultado que comiencen con #
/^$/d	eliminar de líneas vacías del resultado
s/\([:a-z:]*\):\(.*\) /\2:\1 /	Este argumento marca la primera cadena de caracteres en minúsculas que terminan con dos puntos y luego marca la cadena de caracteres que le sigue a los dos puntos. Para el resultado, estas cadenas marcadas cambian de posición.

Ejemplo de sed – agrupar patrones:

```
cat << EOF > sed_chown_example.txt
# use sed to swap the group:owner to owner:group

sudo chown dba:jdoe oraenv.ksh
sudo chown staff:jdoe sysenv.ksh
...
EOF

$ sed '/^#.*$/d;/^$/d;s/\([:a-z:]*\):\(.*\) /\2:\1 /' sed_chown_example.txt
sudo chown jdoe:dba oraenv.ksh
sudo chown jdoe:staff sysenv.ksh
...
```

Usando awk

El programa awk puede ser un manipulador de texto útil – realiza tareas tales como el análisis, el filtrado y el formato sencillo de texto. Toma su entrada de stdin o de archivos y de forma predeterminada, muestra el resultado en stdout. Existe una variedad de releases disponibles para awk con distintos nombres como nawk y gawk. El comportamiento entre las versiones y los release de proveedores de awk varía. awk es distinto a otros comandos revisados en este artículo porque es un lenguaje de programación. Este lenguaje provee funciones internas de cálculo, manipulación de cadenas, control de flujo y formato de texto. Los programadores también pueden definir sus propias funciones creando bibliotecas o funciones definidas por usuario, o scripts autónomos. Dado que awk contiene tantos recursos para demostrar, sólo mostramos unos pocos ejemplos. Por favor, consulte la sección Recursos o las páginas principales para obtener más información.

Al comienzo de este ejemplo se usa awk como filtro para imprimir únicamente sistemas de archivo completo de un sistema Linux. De manera predeterminada, awk utiliza espacio blanco para identificar columnas separadas. El ejemplo examina la columna 5 porque muestra el porcentaje utilizado de espacio en disco. Si la utilización de disco es del 100%, el primer ejemplo imprime el registro en stdout. El argumento siguiente extiende el primero para dar formato a un mensaje, posiblemente para enviar un email o para escribir como parte de un mensaje en un archivo de registro. Se muestra un ejemplo de cómo crear una concordancia utilizando una comparación numérica.

Ejemplo de awk - filtro:

```
$ df –k
Filesystem          1K-blocks      Used Available Use% Mounted on
/dev/sda1           61438632  61381272     57360 100% /
udev                  255788       148    255640   1% /dev
/dev/mapper/datavg     6713132   3584984   3128148  54% /data
rmthost1:/archives/backup    -         -         -   -  /backups
rmthost1:/archives/          -         -         -   -  /amc
rmthost1:/archives/data2     -         -         -   -  /data2

$ df -k |awk '$5 ~ /100%/ {print $0}'
/dev/sda1           61438632  61381272     57360 100% /

$ df -k |awk '$5 ~ /100%/ {printf("full filesystem: %s, mountpoint: %s\n",$6,$1)}'
full filesystem: /, mountpoint: /dev/sda1

$ df -k |awk '$4 > 3000000  {print $0}'
Filesystem          1K-blocks      Used Available Use% Mounted on
/dev/mapper/datavg     6713132   3584984   3128148  54% /data
```

A veces los datos no están delimitados por espacio blanco. Tome, por ejemplo, el archivo /etc/passwd, que está delimitado por el carácter de dos puntos ":". Este ejemplo muestra cómo `awk` utiliza la bandera -F para imprimir el nombre de usuario y el UID de las primeras 5 entradas presentadas en /etc/passwd. Luego, la función `substr()` de awk es demostrada imprimiendo los primeros tres caracteres de la columna 1 del archivo /etc/passwd.

Ejemplo de awk - separador de campo / función de cadena:

```
$ cat /etc/passwd |awk -F: '{printf("%s %s\n", $1,$3)}' |head -5
root 0
daemon 1
bin 2
sys 3
adm 4

cat /etc/passwd |awk -F: '{printf("%s \n", substr($1,1,3))}'|head -5
roo
dae
bin
sys
adm
```

En varias ocasiones, los administradores de sistema o los programadores escriben sus propios scripts `awk` para realizar algún tipo de tarea. Aquí hay un ejemplo de un programa `awk` para obtener el promedio de números encontrados en la tercera columna de un archivo. El cálculo se hace manualmente sumando los datos de la columna 3 en la variable total. NR es una variable interna especial que `awk` usa para hacer seguimiento de cuántos registros fueron procesados. El promedio de la columna 3 se obtiene dividiendo la variable total por NR. El programa muestra los resultados y los datos intermedios para que sea más sencillo seguir la lógica.

Ejemplo de awk – programa / cálculo:

```
cat << EOF > dummy_file2.dat
011  IBM 174.99
012 INTC 22.78
013 SAP 59.37
014 vmw 102.92
EOF

$ cat avg.awk
awk 'BEGIN {total=0;}
        {printf("tot: %.2f arg3: %.2f NR: %d\n",total, $3, NR); total+=$3;}
    END {printf "Total:%.3f Average:%.3f \n",total,total/NR}'

$ cat dummy_file2.dat | avg.awk
tot: 0.00 arg3: 174.99 NR: 1
tot: 174.99 arg3: 22.78 NR: 2
tot: 197.77 arg3: 59.37 NR: 3
tot: 257.14 arg3: 102.92 NR: 4
Total:360.060 Average:90.015
```

Operaciones de cadenas basadas en shell

Shell puede ser un lenguaje de programación poderoso. Al igual que awk, shell ofrece una amplia variedad de opciones para operaciones de cadena, funcionalidad de cálculo, matrices, control de flujo y operaciones de archivo. Debajo hay algunos ejemplos que muestran cómo extraer partes de una cadena de un lado. La operación no cambia el valor de la cadena pero repite cómo debería ser el resultado y frecuentemente se utiliza como asignación de una variable. Use el signo de porcentaje “%” para truncar la derecha del patrón, y use el signo numeral “#” para truncar la izquierda del patrón.

Ejemplo de script shell – extracción de cadena:

```
$ cat string_example1.sh
#!/bin/sh
FILEPATH=/home/w/wyoes/samples/ksh_samples-v1.0.ksh
echo '${FILEPATH}      =' ${FILEPATH}      "  # the full filepath"
echo '${#FILEPATH}     =' ${#FILEPATH}      "  # length of the string"
echo '${FILEPATH%.*}   =' ${FILEPATH%.*}   "  # truncate right of the last dot"
echo '${FILEPATH%%.*}  =' ${FILEPATH%%.*}  "  # truncate right of the first dot"
echo '${FILEPATH%%/w*} =' ${FILEPATH%%/w*} "  # truncate right of the first /w"
echo '${FILEPATH#/*/*/}  =' ${FILEPATH#/*/*/}  "  # truncate left of the third slash"
echo '${FILEPATH##/*/} =' ${FILEPATH##/*/} "  # truncate left of the last slash"

$ ./string_example1.sh
${FILEPATH}=/home/w/wyoes/samples/ksh_samples-v1.0.ksh # the full filepath
${#FILEPATH} = 42                                # length of the string
${FILEPATH%.*}=/home/w/wyoes/samples/ksh_samples-v1.0  # truncate right of the last dot
${FILEPATH%%.*}=/home/w/wyoes/samples/ksh_samples-v1   # truncate right of the first dot
${FILEPATH%%/w*}=/home                           # truncate right of the first /w
${FILEPATH#/*/*/}=wyoes/samples/ksh_samples-v1.0.ksh   # truncate left of the third slash
${FILEPATH##/*/}=ksh_samples-v1.0.ksh            # truncate left of the last slash
```

A modo de ejemplo, supongamos que un administrador necesita cambiar la extensión de un conjunto de archivos .jpg a letras minúsculas. Como los servidores UNIX diferencian mayúsculas y minúsculas, algunas aplicaciones requerirán la extensión en minúsculas, o tal vez el administrador simplemente trate de estandarizar las extensiones de archivo. Cambiar una gran cantidad de archivos a mano o a través de GUI puede llevar horas. A continuación se presenta un script shell que muestra un método para resolver este problema. El ejemplo está compuesto de dos archivos. Primero está setup_files.ksh, que se utiliza para

configurar un árbol de directorio de muestra y poblar el árbol con algunos archivos. También crea una lista de archivos que requieren cambio de extensión. El segundo script llamado fix_extension.ksh lee las listas de archivos, cambiando las extensiones, cuando es apropiado. Como parte del comando `mv` , el operador de cadena % se utiliza para truncar a la derecha del último punto “.” en el nombre de archivo (trunca la extensión). Ambos script también utilizan el comando `find` para mostrar qué se logró después de la ejecución.

Ejemplo de script shell – cambiar extensión de archivo:

```
$ cat setup_files.ksh
mkdir /tmp/mv_demo
[ ! -d /tmp/mv_demo ] && exit

cd /tmp/mv_demo
mkdir tmp JPG 'pictures 1'
touch a.JPG b.jpg c.Jpg d.jPg M.jpG P.jpg JPG_file.JPG JPG.file2.jPg file1.JPG.Jpg 'tmp/
pic 2.Jpg' 10.JPG.bak 'pictures 1/photo.JPG' JPG/readme.txt JPG/sos.JPG

find . -type f|grep -i "\.jpg$" |sort| tee file_list.txt

$ ./setup_files.ksh
./JPG.file2.jPg
./JPG/sos.JPG
./JPG_file.JPG
./M.jpG
./P.jpg
./a.JPG
./b.jpg
./c.Jpg
./d.jPg
./file1.JPG.Jpg
./pictures 1/photo.JPG
./tmp/pic 2.Jpg

$ cd /tmp/mv_demo
$ cat /tmp/fix_extension.ksh
while read f ; do
   mv "${f}" "${f%.*}.jpg"
done < file_list.txt

find . -type f|grep -i "\.jpg$" |sort

$ /tmp/fix_extension.ksh
./JPG.file2.jpg
./JPG/sos.jpg
./JPG_file.jpg
./M.jpg
./P.jpg
./a.jpg
./b.jpg
./c.jpg
./d.jpg
./file1.JPG.jpg
./pictures 1/photo.jpg
./tmp/pic 2.jpg
```

Con el ánimo de crear herramientas útiles y reutilizables, el ejemplo de cambio de extensiones de archivos debería estar más generalizado. Algunas mejoras que me vienen a la mente serían pasar una corriente de nombres de archivo para cambiar, como parte de una tubería. Se podrían agregar banderas de opción para especificar las extensiones de archivo a cambiar (como .mp3 ó .mov), y cómo dar formato a la extensión de archivo, en cuanto a minúsculas, mayúsculas o combinaciones. Las posibilidades están limitadas únicamente por la imaginación y el tiempo del programador.

Resumen

UNIX ofrece una amplia variedad de herramientas para hacer análisis de texto nativamente, en muchos casos, sin la necesidad de depender de intérpretes especiales que pueden no estar instalados. Este artículo es bastante amplio en su sondeo de comandos si se lo compara con la frecuencia de su uso. Los comandos de este artículo están parcialmente demostrados, ya que algunos sistemas implementan banderas o se comportan de forma distinta que otros sistemas. Ciertamente, UNIX ofrece más comandos y formas de lograr las mismas tareas – “Hay más de una forma de hacerlo”.

Sobre el autor

Brad Yoes

es un Migration Engineer de IBM con 15 años de experiencia en migración de aplicaciones, que comenzó a trabajar en UNIX en 1993. Fuera del trabajo disfruta pasar tiempo con su familia, leer literatura documental y correr.