jueves, 16 de noviembre de 2023 20:46

Requisitos previos

Para esta práctica de laboratorio, debe tener un conocimiento sólido de estos comandos de Linux:

- gato
- grep
- cortar

gato:

El comando **cat** nos permite crear uno o varios archivos, ver el contenido de un archivo, concatenar archivos y redirigir la salida en la terminal u otros archivos.

Sintaxis:

```
gato [archivo]
```

grupo:

El comando **grep** , que significa "impresión global de expresión regular", procesa el texto línea por línea e imprime cualquier línea que coincida con un **patrón** específico .

Sintaxis:

```
grep [patrón] [directorio-archivo]
```

Aquí `[file-directory]` está la ruta al directorio/carpeta donde desea realizar una operación de búsqueda. El comando grep también se usa para buscar texto y hacer coincidir una cadena o patrón dentro de un archivo.

Sintaxis:

```
grep [patrón] [ubicación del archivo]
```

cortar:

El comando **de corte** extrae un número determinado de caracteres o columnas de un archivo. Un delimitador es un carácter o conjunto de caracteres que separan cadenas de texto.

Sintaxis:

```
cortar [opciones] [archivo]
```

Para campos separados por delimitadores, se utiliza la opción **-d .** La opción **-f** especifica el campo, un conjunto de campos o un rango de campos que se extraerán.

Sintaxis:

```
cortar -d [delimitador] -f [número de campo]
```

Redirección de E/S de Linux

La redirección se define como cambiar flujos de datos estándar desde una fuente especificada por el usuario o un destino especificado por el usuario. A continuación se muestran los siguientes flujos utilizados en la redirección de E/S:

- Redirección a un archivo usando **>**
- Agregar usando **>>**

Redirección a un archivo

Cada secuencia utiliza comandos de redirección. Se puede utilizar un signo mayor que (**>**) o un signo mayor que doble (**>>) para redirigir la salida estándar.** Si el archivo de destino no existe, se creará un nuevo archivo con el mismo nombre.

Los comandos con un único signo mayor que (**>**) **sobrescriben** el contenido del archivo existente.

```
gato > [archivo]
```

Los comandos con un signo doble mayor que (**>>**) **no sobrescriben** el contenido del archivo existente, pero se agregarán **a** él.

```
gato >> [archivo]
```

Entonces, en lugar de crear un archivo, el comando >> se usa para agregar una palabra o cadena al archivo existente.

Ejercicio

El escenario

Su compañera de trabajo Jane Doe actualmente tiene el nombre de usuario "jane", pero necesita "jdoe" para cumplir con la política de nombres de su empresa. Este cambio de nombre de usuario ya se ha realizado. Sin embargo, algunos archivos que fueron nombrados con el nombre de usuario anterior de Jane, "jane", no se han actualizado. Por ejemplo, "jane_profile_07272018.doc" debe actualizarse a "jdoe_profile_07272018.doc".

Navegue al directorio **de datos** usando el siguiente comando:

```
datos del CD
```

Puede enumerar el contenido del directorio usando el comando **ls** . Este directorio contiene un archivo llamado **list.txt** . También encontrará algunos otros archivos dentro de este directorio.

Para ver el contenido del archivo, utilice el siguiente comando:

```
lista de gatos.txt
```

Producción:

```
001 jane /datos/jane_profile_07272018.doc
002 kwood /datos/kwood_profile_04022017.doc
003 pchow /data/pchow_profile_05152019.doc
004 janez /data/janez_profile_11042019.doc
005 jane /data/jane_pic_07282018.jpg
006 kwood /data/kwood_pic_04032017.jpg
007 pchow /data/pchow_pic_05162019.jpg
008 jane /data/jane_contact_07292018.csv
009 kwood /data/kwood_contact_04042017.csv
010 pchow /data/pchow_contact_05172019.csv
```

Este archivo contiene tres columnas: número de línea, nombre de usuario y ruta completa al archivo.

También puede ver el `/data` directorio completo usando el comando **ls** .

```
es
```

Probemos los comandos que aprendimos en la sección anterior para captar todas las líneas "jane".

```
grep 'jane' ../data/list.txt
```

Esto devuelve todos los archivos con el patrón "jane". También coincide con el archivo que contiene la cadena "janez".

```
001 jane /datos/jane_profile_07272018.doc
004 janez /data/janez_profile_11042019.doc
005 jane /data/jane_pic_07282018.jpg
008 jane /data/jane_contact_07292018.csv
```

Ahora, enumeraremos sólo los archivos que contienen la cadena "jane" y no incluiremos "janez".

```
grep 'jane' ../data/list.txt
```

Esto ahora devuelve sólo archivos que contienen la cadena "jane".

```
001 jane /datos/jane_profile_07272018.doc
005 jane /data/jane_pic_07282018.jpg
008 jane /data/jane_contact_07292018.csv
```

A continuación, usaremos el comando **cortar con el comando grep** . Para el comando de corte, usaremos el carácter de espacio en blanco (' ') como delimitador (indicado por `-d`) ya que las cadenas de texto están separadas por espacios dentro del archivo **list.txt .** También obtendremos resultados especificando los campos usando la opción **-f** .

Busquemos los diferentes campos (columnas) usando `-f` la bandera:

```
grep " jane " ../data/list.txt | cortar -d '' -f 1
```

Producción:

```
001
005
008
```

```
grep " jane " ../data/list.txt | cortar -d '' -f 2
```

Producción:

```
jane
jane
jane
```

```
grep " jane " ../data/list.txt | cortar -d '' -f 3
```

Producción:

```
/datos/jane_profile_07272018.doc
/datos/jane_pic_07282018.jpg
/datos/jane_contact_07292018.csv
```

También puede devolver una variedad de campos juntos usando:

```
grep " jane " ../data/list.txt | cortar -d '' -f 1-3
```

Para devolver un conjunto de campos juntos:

```
grep " jane " ../data/list.txt | corte -d '' -f 1,3
```

Comando de prueba

Ahora usaremos el comando **de prueba** para probar la presencia de un archivo. La **prueba** de comando es una utilidad de línea de comandos en sistemas operativos tipo Unix que evalúa expresiones condicionales.

La sintaxis de este comando es:

```
prueba de EXPRESIÓN
```

Usaremos este comando para verificar si un archivo en particular está presente en el sistema de archivos. Hacemos esto usando el indicador **-e** . Este indicador toma un nombre de archivo como parámetro y devuelve True si el archivo existe.

Comprobaremos la existencia de un archivo llamado **jane_profile_07272018.doc** usando el siguiente comando:

```
si  prueba -e ~/data/jane_profile_07272018.doc; luego  haga eco
de  "El archivo existe" ; else  echo  "El archivo no existe" ;
fi
```

Producción:

```
El archivo existe
```

Crear un archivo usando un operador de redirección

Ahora usaremos el operador de redirección (>) para crear un archivo vacío simplemente especificando el nombre del archivo. La sintaxis para esto es:

```
> [nombre-archivo]
```

Creemos un archivo llamado **test.txt** usando el operador de redirección.

```
> prueba.txt
```

Producción:

```
jane_contact_07292018.csv jane_profile_07272018.doc
janez_profile_11042019.doc kwood_pic_04032017.jpg
kwood_profile_04022017.doc list.txt pchow_pic_05162019.jpg test.txt
```

Para agregar cualquier cadena al archivo **test.txt** , puede usar otro operador de redirección (>>).

```
echo  "Estoy agregando texto a este archivo de prueba" >>
test.txt
```

Puede ver el contenido del archivo en cualquier momento utilizando el comando **cat** .

```
prueba de gato.txt
```

Producción:

```
Estoy agregando texto a este archivo de prueba.
```

Iteración

Otro aspecto importante de un lenguaje de programación es la iteración. La iteración, en términos simples, es la repetición de un conjunto especifico de instrucciones. Es cuando se repite un conjunto de instrucciones varias veces o hasta que se cumple una condición. Y para este proceso, el script bash permite tres declaraciones iterativas diferentes:

- **For** : un bucle for repite la ejecución de un grupo de declaraciones sobre un conjunto de elementos.
- **Mientras** : un bucle while ejecuta un conjunto de instrucciones siempre que la condición de control siga siendo verdadera.
- **Hasta** : un bucle hasta ejecuta un conjunto de instrucciones siempre que la condición de control siga siendo falsa.

Ahora iteremos sobre un conjunto de elementos e imprimamos esos elementos.

```
para yo en 1 2 3; hacer  eco  $i ; hecho
```

Producción:

```
1
2
3
```

Buscar archivos usando el script bash

En esta sección, escribirá un script llamado **findJane.sh** dentro del `scripts` directorio.

Este script debería capturar todas las líneas "jane" y almacenarlas en otro archivo de texto llamado **oldFiles.txt** . Completará el script usando el comando que practicamos en secciones anteriores. No te preocupes, te guiaremos durante todo el proceso.

Navegue al `/scripts` directorio y cree un nuevo archivo llamado **findJane.sh** .

```
cd ~/guiones
```

```
nano findJane.sh
```

Ahora, agrega la línea shebang.

```
bash
 #!/bin/bash
```

Crea el archivo de texto **oldFiles.txt** y asegúrate de que esté vacío. Este archivo **oldFiles.txt** debería guardar archivos con el nombre de usuario " **jane** ".

```
> archivosantiguos.txt
```

Ahora, busque todas las líneas que contengan el nombre " `jane` " y guarde los nombres de los archivos en una variable. Llamemos a esta variable **archivos** , nos referiremos a ella con ese nombre más adelante en la práctica de laboratorio.

Dado que ninguno de los archivos presentes en el archivo **list.txt** está disponible en el sistema de archivos, verifique si los nombres de los archivos presentes en la variable **de archivos** están realmente presentes en el sistema de archivos. Para hacer esto, usaremos el comando **de prueba** que practicamos en la sección anterior.

Ahora, itere sobre la variable **de archivos** y agregue una expresión de prueba dentro del bucle. Si el elemento dentro de la variable archivos pasa la prueba, agréguelo o agréguelo al archivo **oldFiles.txt** .

Una vez que haya terminado de escribir el script bash, guarde el archivo haciendo clic en Ctrl-o, la tecla Intro y Ctrl-x.

Haga que el archivo sea ejecutable usando el siguiente comando:

```
chmod +x encontrarJane.sh
```

Ejecute el script bash **findJane.sh** .

```
./findJane.sh
```

Esto generará un nuevo archivo llamado **oldFiles.txt** , que consta de todos los archivos que contienen el nombre "jane".

Utilice el comando **cat** seguido del nombre del archivo para ver el contenido del archivo recién generado.

```
gato oldFiles.txt
```

Producción:

```
/home/student-02-196e48437fa8/data/jane_profile_07272018.doc
/home/student-02-196e48437fa8/data/jane_contact_07292018.csv
```

Haga clic en *Verificar mi progreso* para verificar el objetivo.

Buscar archivos usando el script bash

comprobar mi progreso

Siga las instrucciones correspondientes para escribir el script bash.

Cambiar el nombre de los archivos usando el script Python

En esta sección, escribirá un script de Python, **changeJane.py** , que toma **oldFiles.txt** como argumento de línea de comando y luego cambia el nombre de los archivos con el nuevo nombre de usuario "jdoe". Completarás el guión, pero te guiaremos a lo largo de

la sección.

Cree un script Python **changeJane.py** en el directorio **/scripts** usando el editor nano.

```
nano cambioJane.py
```

Agrega la línea shebang.

```
#!/usr/bin/env python3
```

Ahora, importe el módulo Python necesario para usarlo en el script Python.

```
importar subproceso de importación sys
```

El módulo sys (parámetros y funciones específicos del sistema) proporciona acceso a algunas variables utilizadas o mantenidas por el intérprete y a funciones que interactúan con el intérprete. El módulo de subproceso le permite generar nuevos procesos, conectarse a sus canales de entrada/salida/error y obtener sus códigos de retorno.

¡Continúa escribiendo el guión para lograr el objetivo!

Dado que **oldFiles.txt** se pasa como argumento de línea de comando, se almacena en la variable sys.argv[1]. Abra el archivo desde el primer argumento para leer su contenido usando el método open(). Puede asignarlo a una variable o utilizar un bloque **with** . Sugerencia: recorra cada línea del archivo usando el método readlines(). Utilice line.strip() para eliminar espacios en blanco o nuevas líneas y recuperar el nombre anterior.

Una vez que tenga el nombre anterior, use la función reemplazar() para reemplazar "jane" con "jdoe". Este método reemplaza las apariciones de cualquier subcadena anterior con la nueva subcadena. Las subcadenas antiguas y nuevas se pasan como parámetros a la función. Por lo tanto, devuelve una cadena donde todas las apariciones de la subcadena anterior se reemplazan con la nueva subcadena.

Sintaxis:

```
cadena.reemplazar (subcadena_antigua, subcadena_nueva)
```

Ahora, invoque un subproceso llamando a la función **run()** . Esta función toma argumentos utilizados para iniciar el proceso. Estos argumentos pueden ser una lista o una cadena.

En este caso, debe pasar una lista que contenga el comando que se ejecutará, seguido de los argumentos del comando.

Utilice el comando **mv** para cambiar el nombre de los archivos en el sistema de archivos. Este comando mueve un archivo o directorio. Toma el archivo/directorio de origen y el archivo/directorio de destino como parámetros. Moveremos el archivo con el nombre anterior al mismo directorio pero con un nombre nuevo.

Sintaxis:

```
destino fuente mv
```

Ahora debe quedar claro. Debe pasar una lista que consta del comando **mv** , seguido de la variable que almacena el nombre antiguo y el nombre nuevo respectivamente a la función **run()** dentro del módulo de subproceso.

Cierra el archivo que se abrió al principio.

```
f.cerrar()
```

Haga que el archivo sea ejecutable usando el siguiente comando:

```
chmod +x cambioJane.py
```

Ejecute el script y pase el archivo **oldFiles.txt** como argumento de línea de comando.

```
./changeJane.py archivosantiguos.txt
```

Navegue hasta el directorio **/data** y use el comando **ls** para ver los archivos renombrados.

```
cd ~/datos
```

```
es
```

```
janez_profile_11042019.doc jdoe_contact_07292018.csv
jdoe_profile_07272018.doc kwood_pic_04032017.jpg
kwood_profile_04022017.doc list.txt pchow_pic_05162019.jpg test.txt
```

Haga clic en *Verificar mi progreso* para verificar el objetivo.