Uso de bash (parte 3)

Expresiones Regulares (REGEX)

Las expresiones regulares son patrones utilizados para buscar y manipular textos. En Bash, las expresiones regulares se utilizan para **realizar coincidencias** de patrones en cadenas de texto, siendo una herramienta más completa y avanzada que los metacaracteres wildcards.

Se componen de una **serie de caracteres (literales y especiales)** que representan el patrón que se desea buscar en un texto.

- Los **caracteres literales** indican búsquedas de coincidencias exactas (los caracteres alfanuméricos son sus propias regex).
  Ejemplo: regex `"hola"` coincidirá con la cadena literal “hola”
- Los **caracteres especiales** (operadores) tienen significados específicos y se utilizan para crear patrones complejos.
  Ejemplo: regex `"[0-9]"` coincidirá con una cadena que contenga un carácter numérico.

La concatenación de regex permite búsquedas de patrones concatenados.
Ejemplo: regex `"[0-9][a-z]"` coincidirá con cadenas que tengan un número seguido de una letra minúscula.

El operador **|** permite búsquedas alternativas entre dos regex.
Por ejemplo: `"foo|bar"` coincidirá con cadenas que contengan “foo” o “bar”.

Metacaracteres de Bash tienen otro significado al usarse en regex.

Expresiones Regulares (REGEX)

Carácter especial punto: .

Permite indicar coincidencia con cualquier carácter, exceptuando salto de línea.

Contenedores: [ ]

- Permite indicar un conjunto de caracteres para búsqueda en un lugar determinado.
  Ejemplo: regex `"[abc]"` coincide con cadenas que tengan ‘a’, ‘b’ ó ‘c’.
- Con `^` se puede indicar la negación de los caracteres del contenedor en la búsqueda.
  Ejemplo: regex `"[^a]"` coincide con cadenas que no tengan ‘a’.
- Con guión - se puede especificar un rango de caracteres.
  Ejemplo: regex `"[0-9A-Za-z]"` coincide con cualquier carácter alfanumérico.

P: Utilizar la página https://regex101.com/ para probar regex.

Expresiones Regulares (REGEX)

Cuantificadores

Permiten controlar el número de coincidencias deseadas para un patrón.

- `?` Representa cero o una ocurrencia del carácter o patrón que lo precede.
- `*` Representa cero o más ocurrencias del carácter o patrón que lo precede.
- `+` Representa una o más ocurrencias del carácter o patrón que lo precede.
- `{n}` Representa n ocurrencias del carácter o patrón que lo precede.
- `{n,m}` Representa entre n y m ocurrencias del carácter o patrón que lo precede.
- `{n,}` Representa al menos n ocurrencias del carácter o patrón que lo precede.

Expresiones Regulares (REGEX)

Puntos de Anclaje

Permiten controlar el lugar de búsqueda de las coincidencias deseadas.

- `^` Representa el inicio de la línea.
- `$` Representa fin de la línea.
- `\<` Representa el principio de la palabra.
- `\>` Representa el fin de la palabra.
- `\b` Representa el límite de la palabra.

No disponibles en todos los softwares

Varios de estos comandos pertenecen a la versión GNU de las regex, conocida como **Expresiones Regulares Extendidas**.

Agrupaciones

Utilizando los paréntesis redondos `( )` se pueden agrupar patrones para trabajarlos de forma conjunta.

grep

Software que filtra líneas de archivos que tengan coincidencia con un patrón (regex). Por ejemplo:

```
> printf "123a\na123\na12b\n567\n" > cadenas.txt
```

```
> grep "^[0-9]" cadenas.txt
```

Es conveniente indicar el patrón entre comillas `"`.

P: Crear regex para: cada cadena del archivo, dos de ellas, tres de ellas y todas.

Algunos argumentos opcionales importantes de grep:

- `-i` no distinguir entre mayúsculas y minúsculas
- `-v` búsqueda inversa
- `-w` búsqueda de palabras completas (no solo subcadenas)
- `-n` mostrar número de línea en output
- `-B , -A` para imprimir líneas antes y después de la coincidencia

Para forzar uso de regex Extendidas usar la opción `-E`.

P: Usar grep y /proc/cpuinfo para extraer información del CPU de su máquina, incluyendo modelo y cores.

sed (stream editor)

Se usa para realizar transformaciones de texto básicas en un texto de input (que puede ser un archivo o de una tubería). Realiza solo una pasada por el input, de modo que es altamente eficiente. También puede ser combinado con otras herramientas de Bash. Su sintaxis es:

```
> sed [opciones] [instrucción] [archivo]
```

- Por defecto sed envía el **texto procesado a stdout** (útil para conectar con otra tubería).
- Para **modificar directamente** un archivo, se puede usar la **opción** `-i`.
- Para suprimir la salida por stdout, se puede usar la opción `-n` (la cual debe ser combinada con otros comandos dentro de la cadena con la “instrucción”.
- **Instrucción:** es una cadena con un formato particular aceptado por sed, la cual acepta algunos **comandos** de un carácter, **regex** y **subcadenas**, todo encerrado entre comillas.

sed (stream editor)

Cadenas de Instrucción

La más utilizada es “buscar y reemplazar” que tiene el siguiente formato:

```
's/regex/reemplazo/'
```

donde `s` es el comando para “buscar y reemplazar” y los slash `/` delimitan los campos del comando (que pueden ser reemplazados por otros caracteres especiales como `;` si es necesario). Ejemplo:

```
> sed 's/1/A/' cadenas.txt
```

```
> echo "114241171" | sed 's/1/A/'
```

Para extender el reemplazo a todas las coincidencias de cada línea, se indica con:

```
's/regex/reemplazo/g'
```

Donde el comando `g` indica reemplazo “global”.

P: Probar sed utilizando las diferentes regex del ejemplo de grep.

Para usar regex Extendidas (como grep) usar opción -E.

sed (stream editor)

Cadenas de Instrucción

Otras instrucciones que puede ejecutar sed:

- Eliminar líneas que coinciden con una regex, comando `d`:

```
> sed '/regex/d' [archivo]
```

- Insertar texto después (antes) de una línea que coincide con una regex, comando `a` (`i`):

```
> sed '/regex/a\texto_a_insertar' [archivo]
> sed '/regex/i\texto_a_insertar' [archivo]
```

- Imprimir líneas que coinciden con una regex, comando `p` (debe combinarse con `-n`):

```
> sed -n '/regex/p' [archivo]
```

También se puede utilizar para imprimir líneas específicas: las instrucciones `'Np'`, `'Np;Mp'` y `'N,Mp'` imprimirán la línea N, la N y la M, y las líneas desde N a M, respectivamente.

P: Probar estas instrucciones con el archivo de prueba cadenas.txt

awk

Es una herramienta de procesamiento de texto que se utiliza para buscar, filtrar y manipular texto (que puede ser un archivo o de una tubería), incluyendo operaciones matemáticas. Su sintaxis básica es similar a sed:

```
> awk [opciones] [instrucción] [archivo]
```

Awk, al tener más funcionalidades de procesamiento, incorpora **su propio lenguaje de programación** para indicar la instrucción a realizar. Estas se componen de bloques de la forma

```
'patrón {acción}'
```

donde ‘patrón’ permite controlar la condición a aplicar, mientras que ‘acción’ ejecuta las operaciones deseadas.

Ejemplo: Visualización de columnas 2 y 3 de un archivo.

```
> awk '{print $2, $3}' archivo.txt
```

Aquí `$2` y `$3` corresponden a variables que almacenan los campos que awk identifica como columnas 2 y 3, de modo que `$0` almacena la línea completa. El delimitador de los campos se puede controlar con la opción `-F`.

P: Descargar archivo croton*.tar.gz del repositorio del curso, y usar awk para filtrar las columnas de masa y filtros de las galaxias.

awk

Como **‘patrón’** se pueden utilizar condicionales que involucren las columnas. Por ejemplo:

```
> awk '$2 > 0 {print $3, $4}' archivo.txt
```

imprime los valores de las columnas 3 y 4 solo cuando el de la 2 es mayor que cero.

‘patrón’ también acepta las instrucciones BEGIN y END para dar instrucciones al inicio y al final del su procesamiento.

También se pueden usar regex para condicionar la ejecución a solo las líneas que tengan coincidencias. Estas tienen que estar contenidas con slash /. Por ejemplo:

```
> awk '/error/ { print $0 }' archivo.log
```

Muestra todas las líneas que contengan “error”.

awk

- Es posible utilizar variables dentro de awk. Por ejemplo:

```
> awk 'BEGIN { i=0 } { i++ } END { print "Tiene" i "líneas" }' archivo.txt
```

Contará todas las líneas del archivo y lo mostrará por pantalla.

- Awk contiene las variables pre-definidas NR y NF que registran el número de filas y columnas durante el procesamiento. Por ejemplo:

```
> awk '{print "Fila:", NR, "Columnas:", NF}' datos.txt
```

Entregará el detalle del número de campos del archivo, por cada fila.

- También permite realizar operaciones matemáticas entre campos. Por ejemplo:

```
> awk '{print ($2+$3)/$4}' archivo.txt
```

Imprimirá, para cada columna, el valor de la operación indicada.

awk

El lenguaje de programación de awk tiene bastantes instrucciones posibles, entre las que se destacan: condicionales múltiples, iteraciones, búsqueda por regex, variables, etc, así como herramientas avanzadas de formateo de texto con **printf**.

Ejemplo:

```
> awk '{for (i=1;i<=NF;i++) sum[i]+=$i} END {for (i=1;i<=NF;i++) print "Columna " i ": " sum[i]}' datos.txt
```

Imprimirá la suma de cada columna del archivo.

Acepta también múltiples acciones separadas por `;`, y permite agrupar instrucciones con el uso de llaves `{ }`.

P: Probar las herramientas de awk con el archivo de galaxias descargado.
P: Investigar sobre instrucciones de awk que no se han abordado en cátedra.

Problemas de práctica

- Utilice el archivo de datos de galaxias simuladas compartido en el directorio del curso, y utilice Bash para:

  - Generar un nuevo archivo que guarde únicamente las galaxias con masa estelar mayor a 10^{10,11,12} Msun/h, que incluya solos las columnas de masa y magnitudes totales.

  - Contar el número de galaxias que hay en intervalos de 0.5 dex en masa estelar.

  - Verificar si el valor de la masa estelar se puede recuperar con las sumas de otras columnas de sus componentes.

- Descargue el sample de resultados de la simulación MDPL2 compartida por el profesor. Desarrolle un script que genere una lista con los redshifts y factor de escala de cada instantánea de la simulación (snapshot), a partir de la información contenida en los nombres de los archivos de la simulación.

Problemas de práctica

- Escriba un script que muestre, para un directorio entregado como argumento, todos los archivos ocultos (que comiencen con .), e indique: su nombre, usuario propietario y permisos del mismo.
- Escriba un script que, dado un directorio entregado como argumento, liste únicamente los archivos que fueron modificados el mismo día, indicando: nombre y fecha/hora de modificación.
- Escriba un script que, dado un directorio entregado como argumento, liste todos los archivos encontrados y los clasifique según su extensión. Esto se debe hacer de forma recursiva en los directorios encontrados, pero la lista mostrada los debe excluir (solo archivos). Incluya un argumento opcional que controle el nivel de profundidad de la recursividad.