Programación para Física y Astronomía

Departamento de Física.

Profesores Claudia Loyola / Alejandro Llanquihuen / Joaquín Peralta

Primer Semestre / 2022

Universidad Andrés Bello

Universidad Andrés Bello®

python powered

Resumen

BASH, comandos fundamentales

- Elementos Básicos de BASH

El intérprete Python

- Variables, asignaciones y tipos
- Declaraciones de entrada y salida (I/O)
- Aritmética
- Trucos en Python

BASH, comandos fundamentales

¿Qué es BASH?

- Bash es una *shell* interactiva, que se carga por defecto en la mayoría de las distribuciones GNU/Linux basadas en Debian.
- Existen otras *shell*, tales como csh o tcsh, sin embargo pondremos especial énfasis en bash, porque es común encontrarla.
- Al abrir un terminal, este siempre abre algún shell, cuando es *bash*, suele verse así

Display standard de BASH

username@machine:~$

- El valor *$* indica que está a la espera de recibir instrucciones.

Repasando lo básico

- Algunas instrucciones básicas de bash
  - pwd : Muestra *dónde* estamos.
  - ls : Lista el contenido del directorio.
  - cp : Copia archivos y/o directorios.
  - mkdir : Crea un nuevo directorio.
  - rm : Elimina archivos y/o directorios.
- Si bien, con unos pocos comandos ya podemos realizar operaciones básicas, existe un gran número de comandos disponibles para distintas aplicaciones. Una lista más completa se puede ver acá: https://ss64.com/bash/
- La filosofía GNU/Linux plantea la creación de herramientas pequeñas (comandos) que pueden ser conectadas entre sí para producir herramientas mucho más complejas.
- Para ello existen las tuberías (|) que pueden ser utilizadas para conectar comandos en la terminal.

Tuberías (Pipes)

- El uso de Tuberías en GNU/Linux, es una de las herramientas más complejas y completas que se pueden utilizar.
- Consideremos por ejemplo, un gran número de archivos, digamos algo así como : *`fichero1.dat`*, *`fichero2.dat`*, ... , *`fichero150000.dat`*. Es decir más de 150 mil archivos.
- Supongamos que cada archivo tiene 5 columnas, con números, y que nuestro objetivo es sumar el valor de la columna 5, y que nos interesa obtener la suma de estos 150mil datos.
- ¿Cómo podemos hacerlo?

Tuberías (Pipes)

- Lo primero es pensar en que debemos analizar cada uno de los archivos, para eso *bash* posee herramientas, tales como *sed* o *awk*.
- Para cada archivo de la lista queremos simplemente la suma de la 5ta columna.
  - **Iteramos** sobre cada archivo
  - Por cada archivo **filtramos** (*awk*) la 5ta columna, y la **sumamos** (*awk*).
  - Luego **juntamos** los resultados
- Con estos nuevos datos, el promedio también será directo con *awk*.

Tuberías (Pipes)

La solución (no necesariamente única), sería de la forma:

```
for i in `seq 1 150000`; do
 cat fichero$i.dat | awk '{SUM += $5} END {print SUM}' >> sums-c5.dat
done
cat sums-c5.dat | awk '{ SUM += $1} END { print SUM/150000 }'
```

Nota

Esperamos que al final del curso, puedan armar este tipo de soluciones, no es necesario desmenuzar el procedimiento global ahora. Sino que comprender el razonamiento que nos lleva a la solución.

Algunas cosas útiles de awk

awk es un programa para el manejo de columnas principalmente, sin embargo, su evolución y fácil programación lo hacen el programa ideal para el manejo de archivos. *Awk* es idiomático, y conviene pensarlo siempre así.

Por ejemplo si queremos imprimir sólo las líneas que cumplen cierta condición en un archivo, uno deberá entregar estas opciones a *awk* como :

awk

```
awk 'conditions {actions}' file
```

Así, si un patrón se cumple en alguna fila, podemos verlo con : *awk '/patron/ {print $0}' archivo*.

Adicionalmente *awk* posee valores por defecto, la principal acción por defecto es *print $0*, por lo que el comando anterior puede reducirse a: *awk '/patron/' archivo*.

Algunas cosas útiles de awk

Esta es una lista con algunas cosas útiles de *awk*.

Comando	Utilidad
`awk 'NR % 6'`	Imprime todo excepto líneas divisibles por 6.
`awk 'NR > 5'`	Imprime a partir de la línea 6.
`awk '$2 == "foo"'`	Imprime líneas cuyo segundo campo es foo.
`awk 'NF >= 6'`	Imprime líneas con 6 o más campos.
`awk '/foo/ && /bar/'`	Imprime líneas que contengan /foo/ and /bar/.
`awk '/foo/ && !/bar/'`	Imprime líneas que contengan /foo/ y no /bar/.
`awk '/foo/ || /bar/'`	Imprime líneas que contengan /foo/ o /bar/.
`awk '/foo/,/bar/'`	Imprime desde línea con /foo/ hasta línea con /bar/.
`awk 'NF'`	Imprime sólo líneas no vacias.
`awk 'NF--'`	Remueve el último campo e imprime la línea.
`awk '$0 = NR" "$0'`	Antepone número de línea.

sed

sed es un programa para el manejo de filas, su sintaxis resumida lo hacen el programa ideal para edición *al vuelo*.

El uso general de *sed* es de la forma:

sed

```
sed opciones 'script' archivo
```

Al igual que *awk* sus opciones son muchas, en la tabla a continuación sólo se muestran algunos casos de utilidad.

Algunas cosas útiles de sed

Esta es una lista con algunas cosas útiles de *sed*.

Comando	Utilidad
`sed -n '5,10p' file.txt`	Imprime entre la línea 5 y la 10.
`sed '20,35d' file.txt`	Imprime todo excepto entre la línea 20 y 35.
`sed -n -e '5,7p' -e '10,13p' file.txt`	Imprime líneas 5-7 y 10-13
`sed 's/version/story/g' file.txt`	reemplaza versión por story en el archivo.
`sed 's/  */ /g'`	Reemplaza múltiples espacios por un espacio simple.
`sed '30,40 s/version/story/g' file.txt`	Reemplaza en un rango específico.
`sed G myfile.txt`	Inserta espacios entre líneas.
`sed '$d' file.txt`	Borra la última línea del archivo.
`sed 'nd' file.txt`	Borra la *n*-ésima línea del archivo.

El intérprete Python

El intérprete Python

- Python es un lenguaje de programación interpretado, orientado a objeto y de alto nivel. Es fácil de aprender, simple de usar y muy poderoso.
- La principal ventaja de usar Python es que este nos permite enfocarnos en “pensar como un programador” más bien que en aprender todas las complejidades recónditas de un lenguaje.

Código ”Hola Mundo” en diversos lenguajes

```
#Codigo en C++
#include <iostream>
int main(){
 std::count<<"Hola mundo!";
}
```

```
#Codigo en Python
print("Hola mundo")
```

```
using System;
namespace HelloWorld{
class Hello {
static void Main(){
Console.WriteLine("Hola Mundo!");
Console.WriteLine("Press Enter");
Console.ReadKey();
    }
  }
}
```

Ambientes de desarrollo

- Un programa en Python es una lista de instrucciones (mezcla de palabras en inglés y matemática) que se denomina código.
- El desarrollo de un programa normalmente requiere de un ambiente de desarrollo. En Python los IDE más comunes son Jupyter, Visual Studio Code, Google Colab, Cocalc, entre otros.
- Sin embargo, en este curso nosotros usaremos la terminal de Linux y el editor Vim para programar y realizar la ejecución de nuestros programas. Dado a grandes ventajas comparativas en termino de acceso a clusters de cómputo remoto.

Ambientes de desarrollo

- Por convención, el nombre de los programas en Python deben terminar con ".py", ejemplo: *programa1.py*.
- La primera línea del fichero que contiene nuestro programa debe indicar el intérprete, en este caso Python. Si bien esto no es obligatorio, es recomendado.

Example (especificando el intérprete en el archivo programa1.py)

```
1 #!/usr/bin/python
2 x=1
3 print(x)
```

- Para ejecutar el programa :

Example (ejecutando el programa en la CLI)

```
>$ ./programa1.py
```

Ambientes de desarrollo

- Si no se indica el intérprete, el programa debe ser ejecutado usando el Python disponible en la línea de comandos.

Example (especificando el intérprete en el archivo programa1.py)

```
>$ python programa1.py
```

Variables, asignaciones y tipos

- Las variables son cantidades de interés en un programa (en física usualmente números, vectores, matrices).
- Las variables reciben un valor mediante una declaración de asignación.

Example (declaración de asignación)

$x = 1$

- Respecto de los nombres de las variables:
  - Tan largos como se desee.
  - Pueden contener letras, números y también el símbolo "_".
  - No pueden comenzar con un número, ni contener cualquier otro símbolo o espacios.
  - Letras mayúsculas y minúsculas son distintas. Esto significa que X y x son dos variables diferentes.
- Existen más de treinta palabras claves en Python que no pueden ser usadas como nombres de variables.
  - if, else, while, print, class, import, etc.

Variables, asignaciones y tipos

- Las variables pueden ser de varios tipos: enteros, flotantes, complejos y string.

Variables

Asignación	Tipo
`x=1`	entero
`x=1.0`	flotante
`x=1.5`	flotante
`x=1.2e34`	flotante usando notación científica
`x=float(1)`	flotante
`x=1+2j`	complejo
`x=complex(1.5)`	complejo
`x="esto es un string"`	string
`x="1.234"`	string

Variables, asignaciones y tipos

- Los espacios entre partes de una declaración no tienen ningún efecto y pueden ser útiles cuando se trabaja con expresiones complejas.

Example

`x=1` y `x = 1` son equivalentes.

Declaraciones de entrada y salida (I/O)

Example (declaración de salida `print( )`)

```
1 #!/usr/bin/python
2 x=1
3 print(x)
```

La declaración ***`print(x)`*** indica al computador imprimir el valor de la variable x en pantalla (salida estándar).

Example (ejecución del código anterior)

```
>$ ./programa2.py
1
>$
```

Declaraciones de entrada y salida (I/O)

La declaración *print()* siempre imprime el valor actual de la variable al momento de la ejecución de la declaración.

Example (otro código)

```
1 #!/usr/bin/python
2 x=1
3 print(x)
4 x=2
5 print(x)
```

Example (salida del código)

```
>$ ./program3.py
1
2
>$
```

Declaración de salida `print()`

- Cada declaración de *`print()`* comienza imprimiendo en una nueva línea.
- La declaración *`print()`* puede ser usada para imprimir más de una cosa en una línea. Los argumentos pasados a *`print()`* son separados por comas. El separador por defecto de los valores impresos es un espacio.

Example

```
1 #!/usr/bin/python
2 x=1
3 y=2
4 print("el valor de x es",x,"y el valor de y es",y)
```

Declaración de salida `print()`

- También es posible modificar el separador impreso por defecto usando el argumento *sep="[letras|números|símbolos]"*.

Example

```
#!/usr/bin/python
x=1.5
z=2+3j
print(x,z,sep="…")
```

Declaración de entrada `input()`

- La forma básica de una declaración de entrada:

Example

```
1 #!/usr/bin/python
2 x=input("Introduzca el valor de x:")
```

- El valor ingresado es un string. Podemos transformar al tipo que necesitemos.

Example

```
1 #!/usr/bin/python
2 tmp=input("Introduzca el valor de x:")
3 x=float(tmp)
4 print("El valor de x es:",x)
```


Nota: Piense en una forma abreviada del ejemplo anterior.

Aritmética

Operaciones aritméticas básicas

Operación	Descripción
x+y	Suma
x-y	Resta
x*y	Multiplicación
x/y	División
x**y	Potencia

Otras operaciones útiles

Operación	Descripción
x//y	Parte entera (función piso) de dividir x por y.
x%y	Operación módulo, el resto de dividir x por y.
	Nota: la operación modulo no aplica en números complejos.

Reglas aritméticas

- El tipo de resultado de una operación depende del tipo de variable.

Example

x=a+b

1.5+2.3 = flotante

1.5+2 = flotante

1+2 = entero

1+(2+3j) = complejo

- Esta regla aplica para sustracción(-), multiplicación(*), división entera y módulo. Recordar que el resultado final es del mismo tipo que los valores de partida, o del tipo más general si existen más de dos tipos de valores de partida.
- Para la división ordinaria (/), la regla es la misma con la excepción que nunca retorna un entero.

Reglas aritméticas

- Para expresiones más complicadas, como x+2*y-z/3, las operaciones siguen reglas similares al álgebra normal.
- / y * antes que – y +
- Si hay varias / y * en una línea se evalúan de izquierda a derecha.
- Potencias son calculadas antes que todo.
- Se puede usar paréntesis para agrupar.
- Python no resuelve ecuaciones por reordenamiento, Python solo sabe de asignaciones.

Example (Descubra el resultado de las asignaciones)

```
2*x+y
```

```
x=1
x=x+1
```

```
x=0
x=x**2-2
```

Algunos trucos en Python

- Los modificadores en Python permiten hacer cambios a las variables de forma sencilla:

Modificadores

Expresión	Descripción
`x+=1`	adiciona 1 a x y es la forma simple de x=x+1
`x-=4`	sustrae 4 de x
`x*=-2.6`	multiplica x por -2.6
`x/=5y`	divide x por 5 veces y

Algunos trucos en Python

- En Python podemos asignar los valores de dos variables mediante una simple declaración:

Expresión 1	Es equivalente a $\rightarrow$	Expresión 2
`x, y = 1, 2.5`		`x=1` `y=2.5`

- Otros ejemplos:

Example

`x, y = 2*x+1, (x+y)/3`

`x, y = y, x` (swap)

Fin