Apéndice A - Materiales adicionales

Contenidos

Este apéndice cuenta con algunos recursos adicionales, como la línea de comando, `git`, editor de texto, etc.

Contenidos: - Section **??** - Herramientas - Section **??** - Section **??** - Section 2.3

1 Línea de comandos

Como mencionamos anteriormente, se hara uso extenso de la línea de comandos. Aqui hay una lista no exhaustiva de los comandos mas simples y mas usados.

OSX / Linux	Windows	Descripción	Ejemplo	Notas
`pwd`	`cd` (sin argumentos)	Imprime el directorio actual	`~$ pwd/home/djangulo`	-
`cd <dir>`	`cd <dir>`	Cambiar de directorio a `<dir>`	`~$ cd Desktop~/Desktop$ pwd/home/djangulo/Desktop`	`..` hace referencia al directorio de “arriba”: `~$ cd ..`

OSX / Linux	Windows	Descripción	Ejemplo	Notas
`ls`	`dir`	Listar archivos en el directorio actual	`~$ lsDesktop Documents Downloads`	`ls -la` muestra archivos escondidos y los tabula
`man <comando>`	-	Muestra el manual de el comando que le sigue	-	-
`cat <archivo>`	`type <archivo>`	Muestra en pantalla los contenidos de `<archivo>`.	-	-
`grep <busqueda> <archivo`	-	Devuelve las líneas de `<archivo>` que contienen `<busqueda>`	-	-
`rm <archivo>`	`del <archivo>`	Borra el archivo `<archivo>`.	`~$ rm script.py`	PELIGRO: no pasa por bandeja de reciclaje
`rm -r <dir>`	`rmdir <dir>`	Borra el directorio `<dir>`.	`~$ rm -r caedro-python`	PELIGRO: no pasa por bandeja de reciclaje

1.1 "Piping" y redireccíon de comandos

Cada programa en la línea de comandos tiene tres flojos de datos que se conectan con el mismo automáticamente:

- `STDIN (0)` - Entrada estándar (datos alimentados al programa)
- `STDOUT (1)` - Salida estándar (datos impresos por el programa, por defecto hacia la terminal)
- `STDERR (2)` - Error estándar (para mensajes de errores, por defecto hacia la terminal)

Redirección es el medio mediante el cual conectamos estos flujos entre los diferentes programas y comandos.

Notese que la redirección funciona con cualquier programa en la terminal, no solo con los usados en los ejemplos debajo

- `|` tambien conocido como `pipe`, crea un flujo de datos de un comando a otro. Lo que hace es que pasa los resultados de un comando a el operando del siguiente.
- <, <<, > y >> manejan redirección de/hacia archivos (la flecha apunta en la dirección de los datos). Las flechas dobles (<< y >>) anexan al archivo, las flechas únicas (< y >) sobreescriben el archivo completo.

Debajo hay algunos ejemplos:

```
~$ ls
Desktop Documents Downloads
~$ ls > mis_carpetas.txt
~$ cat mis_carpetas.txt
Desktop Documents Downloads
```

Digamos que en el directorio `Documents` tengo solo 2 archivos, `script1.py` y `script2.py`, y el directorio `Pictures` no existe.

```
~$ ls Pictures Documents
ls: cannot access 'Pictures': No such file or directory
Documents:
script1.py script2.py
```

Se puede enviar el stderr y el stdout a diferentes archivos utilizando sus índices

```
~$ ls Pictures Documents 1> stdout.txt 2> stderr.txt
~$ cat stdout.txt
Documents:
script1.py script2.py
~$ cat stderr.txt
ls: cannot access 'Pictures': No such file or directory
```

El "piping" le pasa la salida de un comando a la entrada del otro

```
~$ ls
Desktop Documents Downloads
~$ ls | grep Do
Documents
Downloads
```

No hay limite de "piping" se pueden encadenar cuantos comandos se deseen. Aqui un ejemplo que permite buscar que hace la opción de algun comando, en este ejemplo, la opción `-q` del comando `tail`:

- La opcion `-A <numero` de `grep` le dice a `grep` que entrege `<numero>` líneas despues del resultado.
- Se le pone un `backslash` (\) antes del guión para "escapar" el caracter, sino `grep` entiende que es una opción para el.

```
~$ man tail | cat | grep -A 1 "\-q"
       -q, --quiet, --silent
              never output headers giving file names
```

2 Herramientas

Aquí un breve listado de algunas de las herramientas que usaremos a lo largo del curso, y el uso básico de las mismas.

2.1 Virtualenv

Virtualenv es una herramienta que se utiliza para crear una instalación virtual de python, completamente independiente del `python` que utilice el sistema.

Hay multiples razones para hacer esto, pero entre las más importantes están:

- Evita contaminar la instalación de `python` del sistema operativo.
- El `python` del sistema operativo, requiere de permisos elevados para correr, usted quiere evitar esto a toda costa, ya que el código que se corre con permisos elevados puede dañar su sistema o sus archivos de manera permanente.
- Le da portabilidad a su programa: utilizando herramientas de `pip`, se puede reproducir un virtualenv en segundos.

Si instaló `Python` utilizando las instrucciones de más arriba, debe de tener `pip`, el manejador de paquetes de `Python` instalado, instalar `virtualenv` es tan simple como correr:

```
~$ pip install virtualenv
```

Luego para crear el ambiente virtual, se usa la sintaxis

```
python -m venv <nombre-del-directorio>
```

Nosotros le llamaremos `env`

```
~$ python -m venv env
```

Puede tomar de segundos a minutos en terminar.i

2.1.1 Uso de `virtualenv`

`Virtualenv` es una herramienta poderosa, sin embargo, su uso básico resulta muy simple:

Para activar el ambiente virtual creado:

```
~$ source env/bin/activate
(env) ~$ _
```

Cuando el ambiente virtual está activado, todos los paquetes que se instalen, al igual que cualquier referencia a `python` usada, se referirá al `env` creado.

Lo puede confirmar utilizando el comando `which`

```
(env) ~$ which python
/home/djangulo/env/bin/python
(env) ~$ _
```

Para desactivarlo, con el ambiente activado (`env` detrás de su carpeta actual lo indica) solo escriba `deactivate`

```
(env) ~$ deactivate
~$ _
```

Si ya no desea utilizar más el ambiente virtual, solo debe de borrar la carpeta que lo contiene (no olvide desactivar el ambiente primero si no lo ha hecho)

```
(env) ~$ deactivate
~$ _
```

2.1.2 Portabilidad (ejercicio opcional)

Este es un ejercicio práctico que le enseñará como hacer que su programa sea portable, gracias a `virtualenv` y a `pip`, sin importar en que sistema se instale:

Cree un `virtualenv`, llámele `mi_env`, y actívelo de inmediato

```
~$ python -m venv mi_env
~$ source mi_env/bin/activate
(mi_env) ~$ _
```

Instale una librería cualquiera usando `pip`, digamos, `beautifulsoup4`, que es excelente para procesar páginas de `HTML`.

```
(mi_env) ~$ pip install beautifulsoup4
Collecting beautifulsoup4
  Downloading beautifulsoup4-4.8.2-py3-none-any.whl (106 kB)
     |□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□| 106 kB 883 kB/s
Collecting soupsieve>=1.2
  Downloading soupsieve-1.9.5-py2.py3-none-any.whl (33 kB)
Installing collected packages: soupsieve, beautifulsoup4
Successfully installed beautifulsoup4-4.8.2 soupsieve-1.9.5
(mi_env) ~$ pip install beautifulsoup4
```

Luego, guarde los contenidos de sus paquetes instalados con `pip freeze`

```
(mi_env) ~$ pip freeze > requirements.txt
```

Puede inspeccionar los contenidos de `requirements.txt` con `cat`

```
(mi_env) ~$ cat requirements.txt
beautifulsoup4==4.8.2
soupsieve==1.9.5
(mi_env) ~$ _
```

Puede desactivar y borrar su `virtualenv`.

```
(mi_env) ~$ deactivate
~$ rm -rf mi_env
~$ _
```

Ahora digamos que usted, movio su programa a otro computador, pero no tiene su `virtualenv`. Normalmente, usted tendría que recordar todos los paquetes que le instalo a su ambiente virtual, pero gracias al archivo `requirements.txt` que creamos, es tan simple como crear un `virtualenv` nuevo y correr un simple comando de `pip`

```
~$ python -m mi_nuevo_env
~$ source mi_nuevo_env/bin/activate
(mi_nuevo_env) ~$ pip install -r requirements.txt
```

Esto se encarga de instalar todos los paquetes descritos en `requirements.txt`.

A pesar de que esto fue un simple ejercicio (`requieremnts.txt` sólo tenía 2 paquetes, despues de todo), este mismo proceso se puede extrapolar para librerías complejas con cientos o miles de dependencias.

De hecho, esto es lo que utilizan los profesionales de `python` al trabajar en código.

2.2 Editor de texto

Para escribir los scripts, resulta muy útil tener a mano un editor de texto que provea:

- Intellisense: le completa los comandos
- Resaltado de sintaxis: ayuda a detectar errores
- Análisis estático (del código): le indica si hay un problema en su código

Puede utilizar el que desee, pero recomiendo `Visual Studio Code` https://code.visualstudio.com/#alt-downloads, con el plugin oficial de `Python`: https://marketplace.visualstudio.com/items?itemName=ms-python.python.

Una vez instalado, en el menu, seleccione “File”>“Open Workspace” y seleccione la carpeta `introduccion-a-python` creada al principio de esta sección.

Otras opciones: - Visual Studio Code https://code.visualstudio.com/ - Sublime Text https://www.sublimetext.com/ - Xcode https://developer.apple.com/xcode/ - Notepad++ https://notepad-plus-plus.org/

2.3 `git`

Git es un sistema de control de versiones distribuido, que se utiliza para colaboración en diferentes tipos de proyectos.

El uso que le daremos a `git` es sumamente básico, sin embargo, les exhorto a que lo exploren un poco más, ya que le puede ayudar bastante en sus propios proyectos.

Nosotros lo usaremos para:

- Mantener actualizadas las lecciones: estos documentos estarán sujetos a cambios, es importante que mantengan la versión actualizada.
- Descargar los scripts de prueba para los ejercicios.

2.4 Instalación de `git`

Puede descargarlo de https://git-scm.com/downloads.

2.4.1 Mac OS X (>10.9)

```
~$ brew install git
```

- Instale `Homebrew`, si ya no lo ha instalado.

  ```
  ~$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/instal
  ```

- Luego instale `python3` usando `Homebrew`:

  ```
  ~$ brew install git
  ```

2.4.2 Windows

Descarge e instale `chocolatey` de https://chocolatey.org/install#individual, luego, desde `powershell`:

```
choco install git
```

Alternativamente, puede descargarlo from

2.4.3 Linux

- Utilice su package manager para instalar python:
  - Ubuntu / debian:

    ```
    ~$ sudo apt-get update
    ~$ sudo apt-get install git
    ```

  - Arch:

    ```
    ~$ sudo pacman -S git
    ```

  - Fedora linux:

    ```
    ~$ sudo dnf install git
    ```

  - RedHat / Centos / RHEL

    ```
    ~$ sudo yum install git
    ```

2.4.4 Para clonar el repositorio:

```
~$ cd Documents
~/Documents$ git clone https://github.com/djangulo/introduccion-a-python.git
```

Para actualizar el repositorio (en caso de que haya alguna actualización), utilice

```
~$ cd Documents/introduccion-a-python
~/Documents/python-para-arquitectura$ git pull origin main
```