Práctica 1: Administración de Sistemas Operativos e Introducción a Python

Sistemas Distribuios

Curso: 2019/2020

Índice

1. **Investigación sobre comandos en Linux** **2**
2. **Gestión de usuarios** **2**
3. **Introducción a Python** **3**
4. **Gestión del SO usando Python** **3**

Suba un documento, en formato pdf, con las respuestas a las siguientes cuestiones. La práctica 1 se realizará de forma individual. **IMPORTANTE**: no se aceptarán prácticas en formato que no sea PDF! Cualquier detección de copia (de compañeros, de años anteriores, o de internet) implicará suspenso (0) en la práctica.

El objetivo de la práctica consiste en resolver problemas, muchos de los cuales, no se conocen a priori, por lo que es necesario buscar información en Internet, en sitios como `stackoverflow.com` o similares. Se debe indicar en cada respuesta la web donde encontró la información, si ha tenido que buscarla.

1. Investigación sobre comandos en Linux

Responda las siguientes preguntas, investigando en internet:

- Haz un resumen de los distintos comandos y/o herramientas útiles para administrar particiones de discos. ¿Qué recomendaciones acerca del número y tipo de particiones seguiría y porqué?
- Qué variable de entorno posibilita que los comandos básicos, ubicados en /bin, se puedan ejecutar desde cualquier directorio del sistema de ficheros sin tener que indicar la ruta absoluta.
- Tras un cambio en la configuración de red, mediante línea de comandos ¿cómo harías para que los cambios tomen efecto sin necesidad de reiniciar el sistema?
- Comente 3 comandos de superusuario que estime interesantes, indicando utilidad y exponiendo un ejemplo práctico.

2. Gestión de usuarios

Responda a las siguientes preguntas, realizando las operaciones en su máquina y mostrando la salida de las operaciones (por ejemplo, con capturas de pantalla). Busque en Internet si no sabe hacer algo.

- 1. Especifica la lista de órdenes y el resultado obtenido, en la extracción de los siguientes datos:
  - Cómo se llama el usuario que está usando la terminal.
  - En qué grupo estamos.
  - Usuarios conectados al sistema
  - Listar los grupos de nuestro sistema
  - Lista de usuarios que pertenecen al grupo sudo (pista, no hay un comando como tal, hay que mirar en un fichero, ¿en cual?).
- 2.Realiza estas operaciones (indica la orden u órdenes):
  - Crea un usuario llamado sd que tenga como directorio de inicio /home/sd.
  - Muestra el UID que tiene asignado y a qué grupo pertenece inicialmente al ser creado.
  - Crea un grupo llamado grupo_sd
  - Haz que el usuario sd pertenezca al nuevo grupo.
  - ¿Cómo puedes ver/comprobar que pertenece a ambos grupos?
  - Cambia el directorio de inicio del usuario creado, para que pase a ser /home/comunes.

3. Introducción a Python

Partiendo de los conocimientos sobre Python trabajados en el seminario de la asignatura, realice algunas operaciones en Python.

Como referencia de partida, puede consultar, entre otros, los siguientes links:

- http://docs.python.org.ar/tutorial/3/stdlib.html
- https://docs.python.org/2/library/os.html
- https://pymotw.com/2/pwd/
- http://librosweb.es/libro/python/capitulo_10/modulos_de_sistema.html

1. Realizar la interesección de dos listas (nota: los elementos solo se repiten una vez en cada lista).
2. Realizar la unión de dos listas (en la lista final no debe haber elementos repetidos).
3. Crear una función copiar(origen,destino) que copia el contenido del fichero origen, en el fichero destino (usando open()).

4. Gestión del SO usando Python

Realice un estudio sobre ciertos módulos y librerías útiles en la administración del sistema operativo.

Se pide, la realización de los scripts, en código Python, para la acciones indicadas en cada uno de los siguientes apartados:

1. Recordando la variable de entorno que nos permite ejecutar comandos ubicados en /bin, sin necesidad de escribir la ruta completa, ni situarnos en el directorio en cuestión, muestra en pantalla el valor que contiene. Nota: existe un módulo que permite ver esto (usar `import nombreDelModulo` al principio del script)
2. Crea una copia del fichero de configuración de red de Ubuntu (o la distribución que use) y asígnale a dicha copia el nombre interfaces_bck. ¿Dónde está ese fichero? Puedes usar la función que has hecho antes, o alguna que ya exista en python para copiar ficheros.
3. Utiliza la librería/módulo necesario para comprobar que los ficheros anteriores son iguales. Modifique el fichero interfaces_bck cambiando el nombre de la interfaz, y a continuación,vuelva a ejecutar la comparación. La modificación puede ser manual o mediante código Python.
4. Saca por pantalla el listado de directorios de inicio de los usuarios que hay en el sistema (por ej. /home/root , /home/osboxes, ...). Pista: hay un fichero con esta información. También existe una función muy interesante en python llamada split, que convierte de un string en una lista.

Práctica 2: Programación con Sockets

Sistemas Distribuidos

Curso: 2019/2020

Índice

1. Partes de la práctica a entregar **2**

2. Normas para la realización de la práctica **2**

3. Programa 1: Probar sockets **2**

4. Programa 2: Chat Simple **2**
4.1. Mejoras 2

5. Programa 3: FTP simple **2**
5.1. Mejoras 3

6. Evaluación **3**

7. FECHA LIMITE DE ENTREGA **3**

Este documento indica los requisitos para la Práctica 2.

1. Partes de la práctica a entregar

1. Código fuente
2. Documentación en pdf del trabajo desarrollado

2. Normas para la realización de la práctica

- Realización por parejas (se admiten trabajos individuales, pero no de tres estudiantes).
- Los trabajos deberán ser entregados obligatoriamente antes de la fecha de entrega fijada en la actividad habilitada en el campus virtual.
- Las dos partes del trabajo son obligatorias. Si una de ellas no se realiza el trabajo se considerará no presentado.

3. Programa 1: Probar sockets

El primer programa será implementar un programa cliente/servidor que comunique dos procesos. Probar las diferencias que ocurren si el socket es TCP o UDP. ¿Qué diferencias hay? ¿Qué pasa si el cliente se inicia antes que el servidor en el caso TCP y UDP?

4. Programa 2: Chat Simple

Realizar un chat simple, en principio síncrono: un usuario habla, y se espera a que el otro usuario responda. Investigar qué función de Python hay para leer por teclado. Implementar una versión con TCP y otra con UDP. ¿Qué diferencias hay entre ellas? Describir qué flujo ocurre entre los participantes.

4.1. Mejoras

Este chat se puede mejorar añadiendo algún comando, como por ejemplo "desconectar" que desconecta al usuario y avisa al otro usuario para que cierre su conexión de forma segura. También se puede investigar cómo hacer el chat asíncrono (investigar cómo usar hebras en Python, o la función sendall()).

5. Programa 3: FTP simple

Escribir un programa cliente/servidor en el que el servidor recibe por socket el nombre de un fichero y se lo envía al cliente.

5.1. Mejoras

El programa se puede mejorar por ejemplo enviando comandos como "listar", que listará los ficheros disponibles; enviando mensajes de error cuando no exista el fichero, permitiendo que el cliente también pueda subir sus propios ficheros, etc.

6. Evaluación

La nota de la práctica se evalua de la siguiente forma:

- Ejercicio 1: 20 %
- Ejercicio 2: 40 %
- Ejercicio 3: 40 %

Cada parte seguirá la siguiente rúbrica:

- El programa tiene errores sintácticos o semánticos: 0 puntos
- El programa tiene errores de ejecución: 3 puntos
- El programa funciona sin errores, pero hace lo mínimo y las explicaciones son muy escuetas: 5 puntos
- El programa funciona perfectamente, se ha realizado alguna de las mejoras propuestas: 7,5 puntos
- El programa funciona perfectamente, es avanzado, y se han realizado muchas de las mejoras propuestas: 10 puntos

7. FECHA LIMITE DE ENTREGA

La fecha de entrega y presentación será la especificada en el campus virtual

Práctica 3: Creación de API REST con Bottle

Sistemas Distribuidos

Curso: 2019/2020

Índice

1. Partes de la práctica a entregar **2**

2. Normas para la realización de la práctica **2**

3. Parte 1: Servicio Web REST con Bottle **2**

3.1. Mejoras 3

4. Parte 2: Cliente **3**

5. Evaluación **4**

6. FECHA LIMITE DE ENTREGA **4**

Este documento indica los requisitos para la Práctica 3.

1. Partes de la práctica a entregar

1. Código fuente (.zip)
   - Las prácticas se corregirán en Linux, para aquellos que utilicen un S.O. distinto (MacOS o Windows) se recomienda testear el código en una distribución Linux antes de entregarlo.
2. Memoria descriptiva del trabajo realizado (pdf)
   - Se debe proporcionar una descripción detallada del programa implementado así como de su funcionamiento y ejecución.
   - Cualquier mejora implementada pero no descrita en la memoria no será evaluada.
   - Si se utilizan librearías adicionales, debe indicarse en la memoria, así como la versión de Python utilizada.

2. Normas para la realización de la práctica

- Realización por parejas (se admiten trabajos individuales, pero no de tres estudiantes).
- Los trabajos deberán ser entregados obligatoriamente antes de la fecha de entrega fijada en la actividad habilitada en el campus virtual.
- Las dos partes del trabajo son obligatorias. Si una de ellas no se realiza el trabajo se considerará no presentado.

3. Parte 1: Servicio Web REST con Bottle

En esta práctica vamos a trabajar con el framework Bottle visto en el Seminario 3. El objetivo es crear un servicio web de gestión de habitaciones de un hotel.

El servicio debe poseer una lista de habitaciones y cada habitación poseerá, mínimo, los siguientes atributos:

- Identificador.
- Número de plazas.
- Lista de equipamiento (por ejemplo: armario, aire acondicionado, caja fuerte, escritorio, etc.).
- Ocupada (sí/no).

El servicio deberá implementar, mínimo, los siguientes `endpoints`:

1. Dar de alta una nueva habitación.

2. Modificar los datos de una habitación.
3. Consultar la lista completa de habitaciones.
4. Consultar una habitación mediante identificador.
5. Consultar la lista de habitaciones ocupadas o desocupadas.

Queda a **juicio del estudiante** identificar qué tipo de operaciones `HTTP` debe implementar cada `endpoint` (`PUT`, `POST` o `GET`).

Para los `endpoints` 2, 4 y 5, es **obligatorio** el uso de parámetros en el `path` de la URL.

Aunque hemos visto que REST permite transmitir los datos en cualquier formato, se recomienda utilizar JSON[1], tal y como se ha visto en el seminario.

Se valorará positivamente el manejo adecuados de los errores (parámetros incorrectos, `endpoint` no encontrado, operaciones no permitidas) en el servicio y su notificación al cliente.

3.1. Mejoras

- Crear un `endpoint` para eliminar una habitación mediante una operación HTTP DELETE.
- Hacer que la información que maneja el servicio sea persistente, almacenando la información de las habitaciones en un fichero. [2] [3]
- Crear una clase "`Room`" para facilitar el manejo de la información por parte del servicio. [4]
- Crear los `endpoints` adicionales que el estudiante considere oportuno.

4. Parte 2: Cliente

Crear un cliente que permita a un usuario interactuar con la API desarrollada.

Este cliente, le mostrará un menú al usuario con las posibles operaciones que se podrán realizar (de acuerdo a las implementadas en el servicio), solicitará los parámetros necesarios y realizará la llamada al servicio. Posteriormente, mostrará la información al usuario (ver Figura 1).

[1] https://realpython.com/python-json/
[2] https://www.pythonforbeginners.com/files/reading-and-writing-files-in-python
[3] https://stackabuse.com/reading-and-writing-json-to-a-file-in-python/
[4] https://docs.python.org/3/tutorial/classes.html

```
Elige que opción deseas realizar:
    1. Dar de alta una nueva habitación
    2. Modificar los datos de una habitación
    3. Consultar la lista completa de habitaciones
    4. Consultar una habitación mediante identificador
    5. Consultar la lista de habitaciones ocupadas o desocupadas
    6. Salir

> 4

Introduce el identificador de la habitación:

> 80

Habitación 80:
    - 3 plazas.
    - Equipamiento: TV, Wifi y escritorio.
    - Libre.

Elige que opción deseas realizar:
    ...
```

Figura 1: Ejemplo de funcionamiento del cliente

5. Evaluación

La nota de la práctica se evalúa de la siguiente forma:

- Parte 1: 70 %
- Parte 2: 30 %

Cada parte seguirá la siguiente rúbrica:

- El programa tiene errores sintácticos o semánticos: 0 puntos
- El programa tiene errores de ejecución: 3 puntos
- El programa funciona sin errores, pero hace lo mínimo y las **explicaciones** son muy escuetas: 5 puntos
- El programa funciona perfectamente, se ha realizado alguna de las mejoras propuestas: 7,5 puntos
- El programa funciona perfectamente, es avanzado, y se han realizado muchas de las mejoras propuestas: 10 puntos

6. FECHA LIMITE DE ENTREGA

La fecha de entrega y presentación será la especificada en el Campus Virtual.

Práctica 4: Node-RED

Sistemas Distribuidos

Curso: 2019/2020

Índice

1. Partes de la práctica a entregar **2**

2. Normas para la realización de la práctica **2**

3. Flujo 1: Calculadora de Vectores mediante API REST **2**
- 3.1. Mejoras . . . 3
- 3.2. Enlaces de interés . . . 3

4. Flujo 2: MQTT **3**
- 4.1. Mejoras . . . 4
- 4.2. Enlaces de interés . . . 4

5. Flujo 3: Contributed Nodes **4**
- 5.1. Enlaces de interés . . . 4

6. Evaluación **4**

7. FECHA LIMITE DE ENTREGA **5**

Este documento indica los requisitos para la Práctica 4.

1. Partes de la práctica a entregar

1. Código fuente (ficheros .json de los flujos creados) (.zip)
   - Las prácticas se corregirán en Linux, para aquellos que utilicen un S.O. distinto (MacOS o Windows) se recomienda testear el código en una distribución Linux antes de entregarlo.
2. Memoria descriptiva del trabajo realizado (PDF)
   - Se debe proporcionar una descripción detallada del programa implementado así como de su funcionamiento y ejecución.
   - Cualquier mejora implementada pero no descrita en la memoria no será evaluada.

2. Normas para la realización de la práctica

- Realización por parejas (se admiten trabajos individuales, pero no de tres estudiantes).
- Los trabajos deberán ser entregados obligatoriamente antes de la fecha de entrega fijada en la actividad habilitada en el campus virtual.
- Las dos partes del trabajo son obligatorias. Si una de ellas no se realiza el trabajo se considerará no presentado.

3. Flujo 1: Calculadora de Vectores mediante API REST

Implementar en Node-RED una API REST que ofrezca, al menos, tres `endpoints` para realizar operaciones sobre vectores de enteros. Para ello se será necesario utilzar, entre otros, nodos de tipo ‘‘`http in`’’ y ‘‘`http response`’’.

Los datos se pasarán a los `endpoints` en formato JSON, y las operaciones a implementar serán:

- `/suma`: sumar dos vectores de tres posiciones. Ejemplo de interacción con el `endpoint` mediante `cURL`:

```
$ curl -X GET -d '{"a":[1,2,3],"b":[4,5,6]}'
   -H "Content-type: application/json"
   http://localhost:1880/suma

{"resultado":[5,7,9]}
```

- `/resta`: restar dos vectores de tres posiciones, en el orden vector ‘‘a’’ - vector ‘‘b’’. Ejemplo de interacción con el `endpoint` mediante `cURL`:

```
$ curl -X GET -d '{"a":[1,2,3],"b":[4,5,6]}'
   -H "Content-type: application/json"
   http://localhost:1880/resta

{"resultado":[-3,-3,-3]}
```

- `/multesc`: multiplicar un vector de tres posiciones por un escalar. Ejemplo de interacción con el `endpoint` mediante `cURL`:

```
$ curl -X GET -d '{"a":[1,2,3],"n":5}'
   -H "Content-type: application/json"
   http://localhost:1880/multesc

{"resultado":[5,10,15]}
```

3.1. Mejoras

- Añadir `endpoints` para operaciones adicionales.
- Permitir realizar las operaciones con cualquier longitud de vector.
- Utilizar un nodo de tipo ‘‘`file`’’ donde se mantenga un *log* de todas las peticiones que se reciben. Cada vez que se reciba una petición, se almacenará en el fichero una nueva línea de texto, indicando el tipo de operación solicitada, parámetros y resultado.

3.2. Enlaces de interés

- https://www.w3schools.com/js/js_arrays.asp
- https://www.w3schools.com/js/js_loop_for.asp
- https://www.w3schools.com/js/js_json_arrays.asp

4. Flujo 2: MQTT

Suscribirse a un broker MQTT mediante un nodo de tipo ‘‘`mqtt in`’’.

El broker al que os debéis suscribir está en la dirección https://test.mosquitto.org/, puerto `1883`. Éste es un servidor público y gratuito para realizar pruebas con el protocolo MQTT.

Entre los distintos eventos disponibles, deberéis suscribiros a aquel con topic `/merakimv/Q2GV-Y4R8-5Z3L/light`

El evento nos dará de forma regular la luminosidad en `luxs` que registra una cámara de seguridad. La estructura del evento (payload del mensaje) es de la forma ‘‘`{‘‘lux’’: 80.9}`’’. El payload llegará en formato `string`, deberéis utilizar un nodo de la clase `parser` para transformarlo a un `JavaScript Object` y poder trabajar fácilmente con esta información.

Mediante un nodo `function`, deberéis almacenar el valor máximo registrado de todos los eventos recibidos. Para ello será necesario utilizar el `contexto` de Node-RED.

Cada vez que se reciba un evento, se debe mostrar, por la consola de depuración, tanto el valor de luminosidad recibido como el máximo registrado hasta este momento.

4.1. Mejoras

- Investigar cómo publicar eventos en https://test.mosquitto.org/ y crear un flujo para publicar y recibir dichos eventos. **No publicar información sensible**.

4.2. Enlaces de interés

- https://cookbook.nodered.org/#mqtt
- https://test.mosquitto.org/

5. Flujo 3: Contributed Nodes

Investigar de entre los diferentes nodos contribuidos de la librería de Node-RED, aquel que más os llame la atención.

Describir el porqué en la memoria y mostrar un ejemplo de flujo sencillo.

5.1. Enlaces de interés

- https://flows.nodered.org/search?type=node&sort=downloads

6. Evaluación

La nota de la práctica se evalúa de la siguiente forma:

- Parte 1: 50 %
- Parte 2: 35 %
- Parte 2: 15 %

Cada parte seguirá la siguiente rúbrica:

- El programa tiene errores sintácticos o semánticos: 0 puntos
- El programa tiene errores de ejecución: 3 puntos
- El programa funciona sin errores, pero hace lo mínimo y las **explicaciones** son muy escuetas: 5 puntos

- El programa funciona perfectamente, se ha realizado alguna de las mejoras propuestas: 7,5 puntos
- El programa funciona perfectamente, es avanzado, y se han realizado muchas de las mejoras propuestas: 10 puntos

7. FECHA LIMITE DE ENTREGA

La fecha de entrega y presentación será la especificada en el Campus Virtual.