Laboratorio de administración y gestión de redes y sistemas
Prueba escrita sobre las prácticas. 9 de enero de 2014.
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Instrucciones:

Ejecuta `~mortuno/prepara_examen_lagsr` y comprueba que esto ha creado los ficheros `~/lagsr.enero.14/parte.TULOGIN.py`, y `~/lagsr.enero.14/practico.TULOGIN.txt`, donde contestarás las preguntas. (La cadena TULOGIN representa tu nombre de usuario en el laboratorio)

El enunciado te pedirá que hagas una serie de cosas en tu ordenador, es conveniente que lo hagas realmente, aunque en realidad lo único que importa es lo que escribas en el examen, describiendo qué has hecho. Dicho de otro modo: el ordenador te servirá para comprobar que lo estás haciendo bien, pero **es irrelevante lo que hagas en la shell, lo único que importa es lo que escribas en el examen**

Ejercicio 1 (6 puntos)

Edita el script con el nombre `~/lagsr.enero.14/parte.TULOGIN.py`

de forma que, empleando la orden `uptime`, muestre por salida estándar el tiempo que lleve encendido cada máquina del laboratorio en que trabajas ahora. El script debe funcionar aunque haya máquinas apagadas. El informe estará ordenado alfabéticamente y tendrá un aspecto similar a este:

```
kappa01  12:35
kappa02  12:33
kappa04  2:07
kappa05  2 days
```

Observaciones:

- En el informe, los nombres de máquinas deben aparecer sin dominio
- Es recomendable que reutilices código de tu práctica `~/lagsr/practica02/parte_usuarios.py`
- Para saber qué máquinas están encendidas, es recomendable que reutilices tu script `~/lagsr/practica02/parte02.py`
- El formato de salida de `uptime` cambia ligeramente cuando el tiempo es inferior o superior a 24 horas. Ejemplos:

```
mortuno@gsyc:~$ uptime
 11:20:43 up 24 days,  1:54,  1 user,  load average: 0.34, 0.32, 0.41

mortuno@kappa02:~$ uptime
 11:21:42 up  3:27,  1 user,  load average: 0.00, 0.01, 0.05
```

Para simplificar, toma la cadena entre la palabra `up` y la primera coma. Esto es, para tiempos superiores a 1 día, ignora las horas.

Ejercicio 2 (1 punto)

Indica todos los pasos necesarios para montar por sshfs el directorio `/var/tmp/` de la máquina del laboratorio cuyo número es uno más que el tuyo, en el directorio `/var/tmp/remota` de tu máquina.

Ejemplo: si tu máquina es `kappa07`, monta el directorio `/var/tmp/` de `kappa08` en el directorio `/var/tmp/remota` de `kappa07`

Recuerda que si haces un paso, pero no cuentas en el examen que lo has hecho, es como si no lo hubieras hecho. Pega el resultado de la ejecución, de forma similar a la memoria de prácticas.

Ejercicio 3 (3 puntos)

Indica las órdenes necesarias para hacer lo descrito a continuación, y pega el resultado de la ejecución, de forma similar a la memoria de prácticas. Respeta los convenios habituales en el uso de git.

1. Toma el repositorio `repo02` de tu práctica 4.2.2 y haz todo lo necesario para tener dos repositorios *esclavos* descendientes de él, llamados `~/lagsr.enero.14/repo.a` y `~/lagsr.enero.14/repo.b`

2. Crea en `repo.a` un fichero llamado `examen.txt`, escribe dentro la hora actual, mételo en el repositorio y haz todo lo necesario para propagar este cambio a `repo.b`

3. En `repo.b`, modifica el fichero `examen.txt` poniendo la hora actual (que sea distinta a la anterior, al menos un minuto más). Vuelve a meterlo en el repositorio. No lo propages a `repo.a`

4. Haz que `repo.b` vuelva al estado inmediatamente anterior a la modificación que acabas de hacer en la pregunta 3.3. Consulta el estado del fichero `examen.txt`, comprueba que tiene la hora que escribiste en `repo.a`

5. Haz que `repo.b` vuelva a su estado más reciente, como si nunca hubieras hecho el apartado 3.4. Comprueba que el fichero `examen.txt` tiene el valor esperado.

Recuerda que si haces un paso, pero no cuentas en el examen que lo has hecho, es como si no lo hubieras hecho.

Laboratorio de Administración y Gestión de Sistemas y Redes
Prueba escrita sobre la teoría. 9 de enero de 2014
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Instrucciones:

- Encontrarás en el *home* del puesto del laboratorio el fichero el fichero `~/teoria.tulogin.txt`.

  Cámbiale el nombre, reemplazando *tulogin* por tu verdadero login. Por ejemplo, si eres *jperez*, debes hacer

  ```
  mv teoria.tulogin.txt teoria.jperez.txt
  ```

  Revisa que esté bien hecho. Si te equivocas en este paso tan sencillo, suspenderás.

- Dentro del fichero `teoria.jperez.txt`, escribe tu login, nombre y apellidos y contesta al examen.

Ejercicio 1 (2 puntos)

¿Qué es una *trap* de SNMP? ¿Qué pasaría si no existiesen las traps ni nada parecido?

Respuesta

Es un mensaje asíncrono desde un dispositivo administrado hasta un NMS. Las operaciones *get* y *set* son síncronas, esto es, el NMS hace una solicitud y el dispositivo responde. Mientras que la operación *trap* es asíncrona, la genera el dispositivo cuando sucede *algo importante*, sin que el NMS le *pregunte*. Por tanto, las *traps* resultan similares a las excepciones en los lenguajes de programación

Si no hubiese este tipo de comunicación asíncrona, el NMS no percibiría las situaciones relevantes a menos que preguntase explícitamente por ellas al dispositivo.

Ejercicio 2 (2 puntos)

Supongamos que creamos varios usuarios con la orden `adduser`, y posteriormente, por error, borramos el contenido del directorio `/etc/skel`. ¿Qué podría pasar?

Respuesta

A los usuarios ya creados esto no les afecta, ya tienen su copia de los ficheros de configuración por omisión. El problema lo tendrían los usuarios creados posteriormente, a quienes les faltarían estos ficheros.

Ejercicio 3 (2 puntos)

La transparencia 12 del tema *shell II* dice *un intérprete con bit SUID es muy peligroso, normalmente la activación del SUID en un script no tiene efecto*

Explica esto.

Respuesta

Para el usuario, ejecutar un script o ejecutar un ejecutable parece lo mismo, pero para el sistema es muy distinto. Un ejecutable se ejecuta sin más, pero un script es un fichero de texto plano que no se puede ejecutar por sí mismo. Al intentar ejecutarlo, el sistema operativo comprueba que el fichero no incluye el *número mágico* [1] correspondiente a un ejecutable, pero sí empieza por los *números mágicos* almohadilla admiración, que indican que a continuación el fichero especifica la ubicación de un ejecutable (el intérprete) que es el que realmente se ejecutará (cuando reciba el texto del script)[2].

Por eso no tiene efecto activar el SUID en un script, porque el script no se ejecuta, se ejecuta el intéprete (el intérprete de shell bash, el intérprete de python o el que corresponda)

Los intérpretes normalmente pertencen al usuario root, como la mayoría de ficheros *importantes* del sistema. Si un intérprete perteneciente al root tuviera el bit SUID activo, cualquier script que se ejecutase en él tendría permisos de usuario root, con lo que podría hacer cualquier cosa en el sistema. Esto es muy peligroso. Activar el SUID en un fichero ejecutable *normal* puede tener cierto riesgo, pero muy reducido comparado con el caso anterior, porque podemos saber con bastante aproximación qué va a hacer ese programa. Mientras que un intérprete ejecutará cualquier órden que se le pase, es impredecible.

Ejercicio 4 (4 puntos)

Deseamos que el ejecutable `/usr/local/bin/ups` sea un demonio que monitoriza el sistema de alimentación ininterrumpida del ordenador, y que se ponga en marcha cada vez que se enciende el ordenador. Indica qué ficheros son necesarios y con qué nombre. Describe superficialmente cual será su contenido. Debes seguir todos los convenios del arranque en System V.

Respuesta

Para seguir correctamente todos los convenios, habrá que renombrar `/usr/local/bin/ups` para que pase a llamerse `/usr/local/bin/upsd`

Luego hará falta un script que maneje el demonio. Se llamaría `/etc/init.d/ups`, aceptaría al menos los parámetros `START`, `STOP` y `RELOAD`. También podría aceptar otros como `RESTART` o `STATUS`. El demonio no tiene por qué manejarse de forma estándar, el script es el que recibe órdenes estándar y envía al demonio órdenes que ya no son estándar, el script *conoce* el interfaz particular de cada demonio concreto.

[1] Se llama número mágico a aquel que, por convenio, tiene un significado especial

[2] Si no se indica la ubicación del intérprete, el sistema operativo usará el intérprete por omisión del usuario

En tercer lugar, habrá que hacer que el script se ponga en marcha cuando se encienda el ordenador, y que se detenga al apagarlo. Suponiendo que estemos en una distribución basada en debian, crearemos un fichero dentro del directorio correspondiente al nivel de ejecución por omisión, `/etc/rc2.d`, que será un enlace simbólico apuntando al script descrito en el párrafo anterior y cuyo nombre será `SNNups`, donde NN serán dos dígitos que indican el orden dentro de ese nivel, por ejemplo, `S99ups`.

También será necesario detener el servicio en los niveles 0 y 6 (*Halt* y *Reboot*, respectivamente). Para ello, en los directorios `/etc/rc0.d` y `/etc/rc6.d` crearemos enlaces simbólicos apuntando al script, cuyo nombre será `KNNups`, donde `NN` representan dos dígitos, por ejemplo `K99ups`.

Laboratorio de administración y gestión de redes y sistemas
Prueba escrita sobre las prácticas. 17 de junio de 2014.
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Instrucciones:

Ejecuta `~mortuno/prepara_examen_lagrs` y comprueba que esto ha creado los ficheros `~/lagrs.junio.14/mayorfichero.TULOGIN.py`, y `~/lagrs.junio.14/practico.TULOGIN.txt`, donde contestarás las preguntas. (La cadena TULOGIN representa tu nombre de usuario en el laboratorio)

El enunciado te pedirá que hagas una serie de cosas en tu ordenador, es conveniente que lo hagas realmente, aunque en realidad lo único que importa es lo que escribas en el examen, describiendo qué has hecho. Dicho de otro modo: el ordenador te servirá para comprobar que lo estás haciendo bien, pero **es irrelevante lo que hagas en la shell, lo único que importa es lo que escribas en el examen**

Ejercicio 1 (5 puntos)

Edita el script con el nombre `~/lagrs.junio.14/mayorfichero.TULOGIN.py`

de forma que, tomando como base la salida de la orden de shell `ls -l`, muestre el nombre y el tamaño del fichero de mayor tamaño del directorio actual. El nombre del fichero debe incluir el *path* completo.

La salida tendrá un aspecto similar a este:

```
/home/al-05-06/jperez/lagrs.junio.14/practico.jperez.txt    4823
```

Observaciones:

- Obviamente, el script debe funcionar correctamente en cualquier directorio, no solo en el del examen
- Es recomendable que reutilices código procedente de tu práctica `mi_ps.py`

Ejercicio 2 (2 puntos)

Monta en un directorio llamado `~/remoto` de tu puesto del laboratorio, el directorio `test` del *home* del usuario *alumno* en la máquina 193.147.71.62. La contraseña es *admin2014*

Escribe todos los pasos que has seguido. Cuando lo hayas hecho, avisa al profesor para que lo vea.

Ejercicio 3 (3 puntos)

Indica las órdenes necesarias para hacer lo descrito a continuación, y pega el resultado de la ejecución, de forma similar a la memoria de prácticas. Respeta los convenios habituales en el uso de git.

1. Deseamos tener un repositorio maestro llamado `~/lagrs.junio.14/maestro.git` y un esclavo suyo llamado `~/lagrs.junio.14/esclavo`

   Dentro del repositorio habrá un fichero llamado `prueba.txt`

   Haz todo lo necesario para tener esto

2. Escribe dentro de `prueba.txt` el texto *hoy es viernes*. Haz un *commit* y propágalo.

3. Modifica el fichero para que diga *hoy es jueves*. Haz un *commit* y propágalo

4. Muestra un listado compacto de los *commit* que tienes

5. Vuelve el sistema al estado que tenías en el paso 2

6. Modifica el fichero para que diga *hoy es viernes 20*. Haz un *commit* y propágalo

Respuesta

```
mkdir abuelo;
mkdir maestro.git;
cd abuelo;
git init;
touch prueba.txt;
git add prueba.txt;
git ca -m "Crea fichero vacío";
cd ..;
git clone --bare abuelo maestro.git;
mkdir esclavo;
git clone maestro.git esclavo;
cd esclavo;
echo "hoy es viernes">prueba.txt;
git ca -m "Añade texto a fichero";
git push;
echo "hoy es jueves">prueba.txt;
git ca -m "Modifica texto en fichero";
git pull;
git log2;
git co e07e9e6;
echo "hoy es viernes 20">prueba.txt;
git ca -m "Añade dia del mes";
git push
```

Laboratorio de Administración y Gestión de Sistemas y Redes
Prueba escrita sobre la teoría. 17 de junio de 2014
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Instrucciones:

- Encontrarás en el *home* del puesto del laboratorio el fichero el fichero `~/teoria.tulogin.txt`.

  Cámbiale el nombre, reemplazando *tulogin* por tu verdadero login. Por ejemplo, si eres *jperez*, debes hacer

  ```
  mv teoria.tulogin.txt teoria.jperez.txt
  ```

  Revisa que esté bien hecho. Si te equivocas en este paso tan sencillo, suspenderás.

- Dentro del fichero `teoria.jperez.txt`, escribe tu login, nombre y apellidos y contesta al examen.

Ejercicio 1 (2.5 puntos)

La transparencia 57 del tema 2, que habla de los enlaces *duros*, dice lo siguiente: *Dado un fichero, se sabe cuántos nombres tiene. Para saber cuáles son sus nombres, habría que buscarlos.*

Explica esta afirmación

Respuesta

Dado un fichero llamado A, se le puede poner otro nombre, por ejemplo B. Decimos que B es un *enlace duro* de A. El nombre B no apunta al nombre A, sino al mismo fichero apuntado por A.

El sistema de ficheros almacena (en el inodo) el número de nombres que tiene cada fichero. Lo podemos ver con la orden `ls -l`, es el número que aparece tras los permisos.

Cada nombre apunta al fichero, pero el fichero no apunta a sus nombres. Esto está respresentado en la figura: hay una flecha desde cada nombre al fichero, pero la flecha va en un solo sentido. Por tanto, a partir del fichero, no es posible llegar a los nombres. El fichero *sabe* que recibe dos flechas, pero no puede saber *de donde vienen*.

Para conocer todos los nombres del fichero A, habría que recorrer todo el árbol, buscando otros nombres que apunten al mismo fichero. Por ejemplo usando la orden `ls -i A`, que nos indica el número de inodo del nombre A. Tendríamos que buscar por todo el árbol otro nombre con el mismo número de inodo, esto es, otro nombre que apunte al mismo fichero.

Ejercicio 2 (2.5 puntos)

¿Para qué sirve el *journal* en un sistema de ficheros?

Respuesta

Para garantizar que las operaciones sean atómicas, esto es, que se hagan por completo o que no se hagan en absoluto, pero que nunca queden incompletas.

Un sistema de ficheros es una estructura bastante compleja. Si hay algún problema en el sistema en mitad de una operación, por ejemplo un apagón, el sistema podría quedar en un estado inconsistente, con una parte de las estructuras actualizadas y otra no. Ejemplo: mover un fichero consiste en copiarlo en la nueva ubicación y borrar el original. Si la operación queda a medias, podríamos tener dos ficheros o ninguno.

Un sistema de ficheros con *journal* lo que hace es almacenar lo que va a hacer, y luego hacerlo realmente. Si algo falla, una herramienta de recuperación puede comparar lo que dice el *journal* con lo que realmente hay en el sistema de ficheros, y si hay alguna inconsistencia, repararla.

Ejercicio 3 (2.5 puntos)

Tenemos un sistema Unix. Queremos instalar cierto software y disponemos de una versión `.tgz` y otra `.deb`.

1. ¿Cuál deberíamos instalar? Si la respuesta es *depende*, ¿de qué depende?
2. ¿Qué pasaría instaláramos *el otro*?

Respuesta

El formato `.tgz` es general para Unix, el `.deb` es específico de Debian y derivados, por tanto:

1. Depende. Si tenemos un Linux Debian o algún otro sistema basado en Debian, como Ubuntu, deberíamos instalar el `.deb`. Pero si se trata de otra variante de Linux, o bien otro Unix que no sea Linux, entonces normalmente deberíamos instalar el `.tgz`, que es más general.
2. Si intentamos instalar un `.deb` en un sistema que no está basado en estos paquetes, simplemente no podremos, no funcionará nada porque no estarán disponibles los gestores de paquetes.

   Pero normalmente sí será posible instalar un `.tgz` en un Debian. No es recomendable porque lo estamos haciendo *de espaldas* al sistema de paquetes, tendremos que resolver a mano la dependencias y las incompatibilidades que pueda haber. Pero normalmente será posible.

Ejercicio 4 (2.5 puntos)

Un compañero que no ha cursado la asignatura te dice lo siguiente:

No se si entiendo bien el protocolo SNMP. Veo que iso.org.dod.internet.mgmt.mib-2.system.sysUpTime es una representación textual que se corresponde con la representación numérica .1.3.6.1.2.1.1.3. Esto es muy parecido a los nombres de dominio y las

direcciones IP, esto es, la representación en modo texto es una dirección web donde yo puedo consultar el valor correspondiente, en este caso el tiempo que lleva una máquina encendida. Y los números serían la dirección IP que me ofrece ese servicio. Pero la secuencia de números no tiene pinta de dirección IPv4. Parece una una dirección IPv6, así que supongo que este es un ejemplo de SNMP funcionando sobre IPv6

¿Qué le respondes?

Respuesta

Es verdad que la idea se parece un poco. Pero esa representación numérica no tiene nada que ver con una dirección IP, ni v4 ni v6. Son dos representaciones del mismo objeto, pero no son direcciones de hosts.

Laboratorio de administración y gestión de redes y sistemas
Prueba escrita sobre las prácticas. 9 de enero de 2015.
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Instrucciones:

Ejecuta `~mortuno/prepara_examen_lagrs` y comprueba que esto ha creado el directorio

`~/lagrs.enero.15`,

y dentro, los ficheros `parte.TULOGIN.py` y `practico.TULOGIN.txt`,

donde contestarás las preguntas. (La cadena TULOGIN representa tu nombre de usuario en el laboratorio)

El enunciado te pedirá que hagas una serie de cosas en tu ordenador, es conveniente que lo hagas realmente, aunque en realidad lo único que importa es lo que escribas en el examen, describiendo qué has hecho. Dicho de otro modo: el ordenador te servirá para comprobar que lo estás haciendo bien, pero **es irrelevante lo que hagas en la shell, lo único que importa es lo que escribas en el examen**

Ejercicio 1 (4 puntos)

Edita el script con el nombre `~/lagrs.enero.15/parte.TULOGIN.py`

de forma que muestre por salida estándar la dirección IP y la dirección MAC de la tarjeta de red de cada máquina que esté funcionando en el laboratorio en que trabajas ahora. Para averiguar la MAC, usa la orden `ifconfig`

El script debe funcionar aunque haya máquinas apagadas. El informe estará ordenado alfabéticamente y tendrá un aspecto similar a este:

```
theta01  193.147.49.143 5c:f9:dd:77:c5:2a
theta02  193.147.49.144 5c:f9:dd:77:1a:c4
theta04  193.147.49.146 5c:f9:dd:77:ba:b4
theta05  193.147.49.147 5c:f9:dd:77:e1:c2
```

Observaciones:

- En el informe, los nombres de máquinas deben aparecer sin dominio
- Es recomendable que reutilices código de tus prácticas
- En el caso de sistemas configurados en inglés, la orden ifconfig muestra la MAC tras la cadena `HWaddr`. En español, este campo está identificado como `direcciónHW`.

Ejercicio 2 (2 punto)

1. Haz que cada vez que se encianda la máquina virtual `pc01`, envíe un paquete de ping a `193.147.71.64` y escriba en `/var/tmp/log` la salida estándar de la orden (de forma que se conserven todas las salidas y no solo la última)

   Hazlo de la forma más sencilla posible, sin respetar el estándar de los niveles de ejecución de System V

2. Describe brévemente qué habría que hacer para cumplir el estándar

Observa que en el apartado 1 tienes que dar una respuesta completa de todas las órdenes necesarias. En el 2, basta con que indiques qué ficheros utilizar, con qué nombre y en qué directorio, con una descripción de su comportamiento. Pero sin detallar su contenido.

Recuerda que no importa lo que hagas en el laboratorio, solo lo que escribas en el examen.

Respuesta

1. Cualquier fichero cuyo nombre empiece por S y que esté en el directorio `/etc/rc2.d` se ejecutará al iniciar la máquina

   Editamos un fichero llamado por ejemplo `/etc/rc2.d/Sping` y cuyo contenido sea

   ```
   ping -c1 193.147.71.64 >> /var/tmp/log
   ```

   Le damos permiso de ejecución

   ```
   chmod+x /etc/rc2.d/Sping
   ```

   Esto basta para que se ejecuta cada vez que se reinicie el sistema, aunque siempre sería mejor añadir al principio `#!/bin/bash`

2. Para seguir el estándar, el fichero debería llamarse `/etc/rc2.d/SNNping` siendo NN un número entre 00 y 99

   Ademas, este fichero no debería tener directamente el contenido del script, debería ser un enlace simbólico a `../init.d/ping`

   A su vez, el fichero `/etc/init.d/ping` debería ser un script que aceptase, al menos, los parámetros `START`, `STOP` y `RELOAD`, y que llamase al verdadero fichero con el código del demonio, p.e. `/usr/local/pingd`

Obsérvese que en ningún caso editamos `bash_profile`, `bashrc` o similares porque la especificación se refiere al inicio de la máquina, no al inicio de la sesión de ningún usuario.

Ejercicio 3 (2 puntos)

Usando `screen`

1. Entra por ssh en otra máquina de este laboratorio
2. Lanza la orden `top` y déjala corriendo
3. Cierra la conexión con la máquina remota, sin que se interrumpa el `top`
4. Vuelve a entrar en la máquina remota y recupera la sesión

Recuerda que no importa lo que hagas en el laboratorio, solo lo que escribas en el examen.

Respuesta

```
ssh theta01
screen
top
(creamos una nueva ventana con ctrl a c)
screen -d  # nos desasociamos. Esto ya se ejecuta en la nueva ventana
screen -ls # paso opcional, para comprobar que estamos desasociados
exit
ssh theta01 # nos reconectamos
screen -ls # vemos el número de sesión, p.e. 1234
screen -r 1234
(cambiamos a la ventana de top con ctrl a, los cursores e intro)
```

Ejercicio 4 (2 puntos)

1. Crea un repositorio git llamado

   `~/lagrs.enero.15/mirepo`

2. Mete dentro el fichero `~/lagrs.enero.15/mirepo/holamundo.txt` cuyo contenido será el texto *feliz 2014*. Etiqueta esto como *primer commit*
3. Haz que el fichero contenga el texto *feliz año*. Etiqueta esto como *segundo commit*
4. Haz todo lo necesario para recuperar el fichero del *primer commit*
5. En el fichero del primer commit, reemplaza el texto por *feliz 2015* y haz un tercer commit etiquetado como *corrige la fecha del primer commit*
6. Muestra un log para comprobar que siguen siendo visibles los tres commits

Recuerda que no importa lo que hagas en el laboratorio, solo lo que escribas en el examen.

Respuesta

```
mkdir mirepo
cd mirepo
git init
echo "feliz 2014">holamundo.txt
git add holamundo.txt
git ca -m "primer commit"
echo "feliz año">holamundo.txt
git ca -m "segundo commit"
git log
git co 9316a16bc1a # Vuelta al "presente"
cp holamundo.txt ..
git co master
cp ../holamundo.txt .
echo "feliz 2015" > holamundo.txt
git ca -m "corrige la fecha del primer commit"
git log
```

Hemos creado el repositorio, el primer commit y el segundo. Cuando volvemos al primer commit, copiamos el fichero en cualquier lugar, por ejemplo en el directorio padre. Tras esto, volvemos a la rama master con la órden `git co master`, lo que informalmente llamamos *volver al presente*. Si omitimos este paso, nos habremos quedado en la estado *detached head*, esto es, un estado sin rama. Y habremos perdido el segundo commit.

Tras recuperar el fichero, creamos el tercer commit y comprobamos que todo está bien.

Laboratorio de Administración y Gestión de Redes y Sistemas
Prueba escrita sobre la teoría. 9 de enero de 2015
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Instrucciones:

- Encontrarás en el *home* del puesto del laboratorio el fichero el fichero `~/teoria.tulogin.txt`.

  Cámbiale el nombre, reemplazando *tulogin* por tu verdadero login. Por ejemplo, si eres *jperez*, debes hacer

  ```
  mv teoria.tulogin.txt teoria.jperez.txt
  ```

  Revisa que esté bien hecho. Si te equivocas en este paso tan sencillo, suspenderás.

- Dentro del fichero `teoria.jperez.txt`, escribe tu login, nombre y apellidos y contesta al examen.

Ejercicio 1 (2.5 puntos)

La transparencia 10 del tema 7 dice: *En Unix y Linux se usan mucho otras herramientas de monitorización como Nagios y Zabbix, que ofrecen sus propios servicios pero también incluyen SNMP*

1. Explica este párrafo. ¿por qué se usan otras herramientas? ¿qué significa que ofrezcan sus servicios? ¿qué significa que incluyan SNMP?

2. ¿Va esto contra el estándar?

3. Ventajas e inconvenientes de SNMP frente a herramientas como Nagios y Zabbix. ¿Cuándo usar uno u otro?

Respuesta

1. SNMP es un protocolo que no ha tenido demasiado éxito, tiene muchas limitaciones. Como todo protocolo, pueden usarse desde muchas herramientas, como Nagios, Zabbix y similares. Por ejemplo, podemos tener una impresora que exporte su estado mediante SNMP, y monitorizarlo desde Zabbix.

   Pero estas herramientas no solo soportan el protocolo estándar de SNMP, sino que también ofrecen sus propios agentes, que usan su propio protocolo, más potente y flexible.

   Si no hablamos de un host sencillo como una impresora, sino de otro más complejo, como un ordenador, también podremos exportar su estado mediante un agente (cliente) SNMP. Pero esto no es muy habitual, posiblemente preferiremos emplear el cliente específico de nuestra herramienta de monitorización.

2. Si solo usamos agentes SNMP, estamos siguiendo el estándar. Pero si se usan agentes particulares (cosa muy frecuente), sí, esto va contra el estándar.

3. SNMP es universal, vamos a encontrar un agente SNMP en cualquier dispositivo con un mínimo de calidad y capaz de conectarse a Internet. Otras herramientas más avanzadas usan sus propios protocolos no universales. Típicamente se usa SNMP cuando no hay otra opción disponible, por ejemplo impresoras, switches, cámaras IP, dispositivos domóticos, etc. En estos dispositivos no vamos a encontrar soporte nativo para, por ejemplo, Zabbix. Cuando se trate de monitorizar ordenadores convencionales, sí podremos usar agentes propios de nuestra herramienta.

Ejercicio 2 (2.5 puntos)

Un servidor de DHCP puede estar configurado en modo *authoritative* o en modo *non authoritative*

1. ¿Qué significa esto?

2. ¿Quién decide que un servidor esté en un modo u otro?

3. ¿Qué garantías tiene un cliente de que un servidor *authoritative* lo es realmente? ¿Cómo evitar que un servidor *non authoritative* conteste como *authoritative*?

4. ¿Qué problemas puede causar un servidor de DHCP que tenga este parámetro mal configurado?

Observación: La transparencia 8 del tema 6 explica el comportamiento de un servidor *authoritative* y *non authoritative* ante un DHCPREQUEST. Pero **no** es eso lo que debes contestar. Cíñete al enunciado del ejercicio.

Respuesta

1. Una respuesta con el flag *authoritative* activado significa que el servidor declara que su respuesta es *oficial*, el servidor está indicando que él es quien tiene autoridad en ese segmento de red.

   Una respuesta *non authoritative*, el servidor no asegura ser la autoridad oficial.

   Cuando un cliente hace una petición *DISCOVERY*, si hay varios servidores recibirá varias respuestas *OFFER*, la norma permite que el cliente elija la respuesta más adecuada, según su criterio. (Debería preferir una con autoridad frente a una sin autoridad, aunque muchas implementaciones se quedan con la primera en llegar)

   Cuando un cliente pide una dirección y al servidor le parece incorrecta, si tiene autoridad se *atreve* a contradecirle. Si el servidor no tiene autoridad, no responde.

2. El administrador de la máquina donde está el servidor de DHCP. Debería ser el administrador de la red, pero no tiene por qué serlo. Es imposible evitar que un servidor se atribuya la autoridad que no tiene.

3. Un servidor de DHCP mal configurado puede ser muy problemático. Supongamos un atacante que instala un servidor en su portátil y lo lleva a una red ajena. El servidor puede empezar a repartir direcciones incorrectas, puede inutilizar la red con parámetros incorrectos, o peor aún, *infectar* la red proporcionando direcciones de servidores con malware.

   Si las repuestas del atacante son *non authoritative* y el servidor de la red víctima también envía respuestas *non authoritative*, los clientes podrían usar indistintamente una u otro. Al igual que el caso en que atacante y víctima declaran ser *authoritative*.

   El peor caso es el de una red víctima donde el servidor tenga la configuración por omisión (*non authoritative*), y el atacante esté dando respuestas (*authoritative*). Las víctimas preferirán al atacante.

   El caso menos dañino es el de un atacante, posiblemente involuntario, que tenga un servidor por omisión (*non authoritative*). Si el administrador de la red ha tenido la precaución de configurar el servidor oficial como (*authoritative*), los clientes preferirán el oficial.

   Esto es lo previsto en el estándar, pero como hemos dicho, muchas implementaciones atienden la primera respuesta recibida, sea cual sea, por lo que aceptar la respuesta correcta o la del atacante acaba resultando aleatorio.

Ejercicio 3 (2.5 puntos)

Probablemente conoces la *flamenca del whatsapp*, un *emoji* (icono) que representa una mujer vestida de rojo, bailando.

¿Sería posible escribir en Linux un texto que la incluya? ¿Qué haría falta?

Respuesta

Unicode es un estándar de codificación de caracteres. Es muy completo, incluye no solo todos los caracteres de prácticamente cualquier lenguaje humano, sino también iconos. Por ejemplo, desde la versión 6.0 (año 2010), incluye la *flamenca*, que es el carácter `'DANCER' (U+1F483)`

Toda herramienta que soporte unicode nos permitirá manejar este y cualquier otro carácter, aunque para poder visualizarlo, será necesario tener instalado el *font* adecuado. Los fonts pueden cambiar de aspecto, por ejemplo este icono en algunos fonts representa un hombre de color amarillo.

Ejercicio 4 (2.5 puntos)

Un script de shell bash normalmente empieza por `#!/bin/bash`

1. ¿Por qué?
2. ¿Qué sucede en caso contrario?

3. ¿Qué sucede con el intento de ejecución de un script sin permiso de ejecución?

Respuesta

1. Para indicar explícitamente donde está el ejecutable que sabe interpretar ese texto. La secuencia almohadilla-admiración es un *número mágico*, un convenio que indica que si estos son los primeros caracteres de un fichero, lo que aparece a continuación es el path del intérprete.

2. Que el sistema no sabría cuál es el intérprete, intentaría ejecutarlo con el intérprete por omisión que haya especificado el usuario. Si ambos coinciden, todo funciona bien. En otro caso, para ejecutarlo sería necesario invocar al intérprete (bash) y luego pasarle el script, bien por la entrada estándar, bien como primer argumento.

3. Depende. Si se intenta ejecuta el script directamente, no funcionará. Pero si se invoca a la shell y se le pasa el script, se ejecutará normalmente.

Laboratorio de administración y gestión de redes y sistemas
Prueba escrita sobre las prácticas. 24 de junio de 2015.
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Instrucciones:

Ejecuta `~mortuno/prepara_examen_lagrs` y comprueba que esto ha creado el directorio

`~/lagrs.junio.15`,

y dentro, los ficheros `claves.tgz`, `info_memoria.TULOGIN.py` y `practico.TULOGIN.txt`,

donde contestarás las preguntas. (La cadena TULOGIN representa tu nombre de usuario en el laboratorio)

El enunciado te pedirá que hagas una serie de cosas en tu ordenador, es conveniente que lo hagas realmente, aunque en realidad lo único que importa es lo que escribas en el examen, describiendo qué has hecho. Dicho de otro modo: el ordenador te servirá para comprobar que lo estás haciendo bien, pero **es irrelevante lo que hagas en la shell, lo único que importa es lo que escribas en el examen**

Ejercicio 1 (4 puntos)

Tomando como punto de partida tu práctica 3.4, escribe el script

`~/lagrs.junio.15/info_memoria.TULOGIN.py` que devolverá el total de memoria instalada (en gigas) y el porcentaje de memoria libre en cada ordenador del laboratorio.

Extrae la información con la orden `free -m`

Ejercicio 2 (3 puntos)

En la práctica 1.16 configuraste tu cuenta para poder entrar desde una máquina del laboratorio hasta otra sin teclear contraseñas.

1. Asegúrate de que esto sigue funcionando correctamente
2. Entra por ssh con el usuario *examen1* en la máquina 193.147.71.62. Lo necesario para hacer esto lo encontrarás en `~/lagrs.junio.15/claves.tgz` (que también contiene ficheros sin ninguna utilidad)
3. Copia el fichero `~/prueba.txt` que encontrarás en la máquina del apartado 2.2 en el directorio `~/lagrs.junio.15/prueba.txt` de tu host
4. Averigua qué codificación emplea
5. Conviértelo a la codificacion por omisión empleada en tu host
6. Asegúrate de que lo que hiciste en el apartado 2.1 sigue funcionando

Ejercicio 3 (3 puntos)

En el directorio `/home/examen2/mail` de la máquina 193.147.71.62 hay un repositorio git. Es accesible para el usuario `examen2`, de contraseña `lagrs.junio.15`

1. Clona este repositorio en el directorio `~/lagrs.junio.15/git` de tu host
2. En principio no habrá ningún fichero en el directorio de trabajo, pero en la revisión etiquetada como *corrige correo*, econtrarás un script llamado `mail_amazon.py`

   Recupéralo
3. Observa que en este script, parece que se declaran las variables `smpt_username` y `smpt_passord` , pero falta el contenido de las variables, no hay nada a la derecha del igual
4. Podrás encontrar estos valores en la revisión etiquetada como *versión inicial*
5. Vuelve a la rama máster
6. Prepara una versión personalizada y completa del script, esto es, rellenando lo que falta en la versión de la etapa 3.3. con los datos de la versión 3.4
7. Añade tu nombre y login en la primera línea del script. Añade al repositorio un fichero llamado `~/leeme.txt` (cuyo contenido puede ser *blablablá*)
8. Crea en tu repositorio local una nueva revisión con todo lo que acabas de hacer, etiquetada como `version_jperez` (donde jperez sería tu verdadero login)

Laboratorio de Administración y Gestión de Redes y Sistemas
Prueba escrita sobre la teoría. 24 de junio de 2015
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Instrucciones:

- Encontrarás en el *home* del puesto del laboratorio el fichero el fichero `~/teoria.tulogin.txt`.

  Cámbiale el nombre, reemplazando *tulogin* por tu verdadero login. Por ejemplo, si eres *jperez*, debes hacer

  ```
  mv teoria.tulogin.txt teoria.jperez.txt
  ```

  Revisa que esté bien hecho. Si te equivocas en este paso tan sencillo, suspenderás.

- Dentro del fichero `teoria.jperez.txt`, escribe tu login, nombre y apellidos y contesta al examen.

Ejercicio 1 (2.5 puntos)

Explica qué sucede al ejecutar en la shell las siguientes órdenes. Cuenta no solo qué sucede, sino cómo sucede, esto es, cómo funcionan en cada ejemplo los argumentos, las redirecciones, entrada estándar, salida estándar, salida de error, pipes, etc

1. `cat`
2. `cat file1 file2 > file3`
3. `cat file1 | less`
4. `cat > file1`

Ejercicio 2 (2.5 puntos)

Supongamos que en nuestro sistema linux hemos montado el directorio raiz en `/dev/sda1`

1. Explica qué significa esto
2. Si ejecutamos `ls /dev/sda1` no vemos los ficheros que contiene `/`.

   Si ejectutamos `cd /dev/sda1` se produce un error

   ¿Por qué?

Ejercicio 3 (2.5 puntos)

Alguien te dice *en mi empresa tenemos un montón de impresoras de diferentes marcas y queremos ahorrar tóner, que las impresoras siempre estén en modo ahorro, a menos que un usuario realmente necesite una calidad más alta. Pero que si el usuario que usó la calidad alta deja esta configuración en el panel de la impresora, algún sistema lo vuelva a configurar en modo ahorro. ¿Se puede hacer esto mediante SNMP? ¿Cómo?*

¿Qué respondes?

Ejercicio 4 (2.5 puntos)

La transparencia 7 del tema 6 dice *El cliente puede renunciar a su lease anticipadamente con un DHCPRELEASE*

¿Qué significa esto? ¿Por qué motivo el cliente haría tal cosa?

Laboratorio de administración y gestión de redes y sistemas
Prueba escrita sobre las prácticas. 18 de diciembre de 2015.
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Instrucciones:

Ejecuta `~mortuno/prepara_examen_lagrs` y comprueba que esto ha creado el directorio

`~/lagrs.enero.16`,

y dentro, los ficheros `claves.tgz`, `info_memoria.TULOGIN.py` y `practico.TULOGIN.txt`,

donde contestarás las preguntas. (La cadena TULOGIN representa tu nombre de usuario en el laboratorio)

El enunciado te pedirá que hagas una serie de cosas en tu ordenador, es conveniente que lo hagas realmente, aunque en realidad lo único que importa es lo que escribas en el examen, describiendo qué has hecho. Dicho de otro modo: el ordenador te servirá para comprobar que lo estás haciendo bien, pero **es irrelevante lo que hagas en la shell, lo único que importa es lo que escribas en el examen**

Ejercicio 1 (3 puntos)

Tomando como punto de partida tu prácticas 3.4 y 3.5, escribe el script

`~/lagrs.enero.16/info_disco.TULOGIN.py`

Que a partir de la orden `df`, mostrará, para cada máquina del laboratorio que ocupas actualmente, el porcentaje de uso de cada sistema de ficheros y la suma total.

De forma similar a esta:

```
delta02  15+0+1+1+0+1+1+21+1+20=61
delta04  19+0+2+1+0+4+1+21+8+20=76
delta05 ...  [etc]
```

El programa debe tolerar que alguna máquina no responda correctamente.

Ejercicio 2 (1.5 puntos)

Edita un fichero de texto llamado

`~/lagrs.enero.16/origenes.txt`

En su interior, escribe un texto que contenga en cada línea un nombre de fichero, incluyendo su path completo. Por ejemplo esto:

```
/tmp/1
/tmp/2
/tmp/3
```

1. Empleando las órdenes xargs y touch, haz que si no existen los ficheros indicados en el fichero de texto origenes.txt, se creen. (Si ya existen no importa lo que suceda con ellos)

2. Empleando las órdenes xargs y touch, haz que en el directorio de trabajo actual se cree un enlace simbólico a cada uno de los ficheros indicados en origenes.txt

respuesta

1. `cat origenes.txt|xargs touch`

2. `cat origenes.txt |xargs -I{} ln -s {} .`

Ejercicio 3 (2.5 puntos)

1. Crea un directorio llamado `~/lagrs.enero.16/base`

2. Escribe los ficheros

   `~/lagrs.enero.16/base/f1.txt`

   `~/lagrs.enero.16/base/f2.txt`

   Contendrán un texto de ejemplo cualquiera, como *este es el fichero f1*

3. Haz todo lo necesario para que `/tmp/r1` y `/tmp/r2` y

   sean repositorios cuyo contenido sea el mismo que `base`

4. Añade un fichero llamado f3.txt al repositorio r1

5. Propaga este cambio a r2

Observaciones:

- Es necesario que respetes las convenciones habituales en git
- Recuerda que no importa lo que hagas en el ordenador del laboratorio, sino lo que escribas en el examen

Respuesta

```
cd   # vamos al directorio home
mkdir lagrs.enero.16/base
cd lagrs.enero.16/base
echo "este es el fichero f1" >  f1
echo "este es el fichero f2" >  f2
git init   # iniciamos el repositorio abuelo
git add f1 f2
git ca -m "commit inicial"
cd ..
```

```
mkdir padre.git
git clone --bare base padre.git # clonamos el abuelo al padre
cd /tmp
mkdir r1 r2
git clone ~/lagrs.enero.16/padre.git r1 # clonamos el hijo r1
git clone ~/lagrs.enero.16/padre.git r2 # clonamos el hijo r2
cd r1
touch f3
git add f3
git ca -m "añade fichero f3"
git pull  # este paso no hace falta, pero es buen hábito
git push # propagamos cambios del nieto al padre
cd ../r2
git pull # propagamos cambios del padre al nieto
git push # este paso no hace falta
```

Ejercicio 4 (2 puntos)

1. Sea TUHOST la máquina del laboratorio que estás usando. Sea OTROHOST una máquina de laboratorio que esté vacía, funcionando y justo a tu derecha. (si no es posible, dile al profesor que te indique cuál es OTROHOST). Indica el nombre de TUHOST y OTROHOST, para que conste en el examen

2. Haz todo lo necesario para que cuando lances romanclient.py contra el puerto 7777 de TUHOST, responda el romanserver.py del puerto 9999 de OTROHOST

Respuesta

```
# Establecemos el túnel
jperez@TUHOST ~/lagrs.enero.16 $ ssh -L 7777:OTROHOST:9999 jperez@OTROHOST

# Desde otra terminal entramos en otrohost y lanzamos romanserver
jperez@TUHOST ~/lagrs.enero.16 $ ssh OTROHOST
jperez@OTROHOST ~ $ cd bin/
jperez@OTROHOST ~/bin $ ./romanserver.py TCP 9999

# Desde otra terminal lanzamos romancliente contra el puerto local
jperez@TUHOST ~ $ cd bin/
jperez@TUHOST ~/bin $ romanclient.py localhost TCP 7777 23
XXIII
```

Ejercicio 5 (1 puntos)

1. Indica qué es necesario para que tu script `aguad` de la práctica 5.3 se ejecute los lunes de septiembre, octubre y noviembre, cada 5 minutos

2. Indica qué es necesario para que ese mismo script se ejecute a las 9 de la mañana, de lunes a viernes

Respuesta

Con la orden `crontab -e` creamos las siguientes tablas:

```
#m   h  dom  mon    dow   command
*/5  *   *   9-11   1     /home/al-12-13/jperez/lagrs/pc01/aguad
```

```
#m   h  dom  mon    dow   command
0    9   *   *      1-5   /home/al-12-13/jperez/lagrs/pc01/aguad
```

Laboratorio de Administración y Gestión de Redes y Sistemas
Prueba escrita sobre la teoría. 18 de diciembre de 2015
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Instrucciones:

- Encontrarás en el *home* del puesto del laboratorio el fichero el fichero `~/teoria.tulogin.txt`.

  Cámbiale el nombre, reemplazando *tulogin* por tu verdadero login. Por ejemplo, si eres *jperez*, debes hacer

  ```
  mv teoria.tulogin.txt teoria.jperez.txt
  ```

  Revisa que esté bien hecho. Si te equivocas en este paso tan sencillo, suspenderás.

- Dentro del fichero `teoria.jperez.txt`, escribe tu login, nombre y apellidos y contesta al examen.

Ejercicio 1 (2.5 puntos)

Un sistema operativo de software libre como Linux ¿se puede vender?

Respuesta

Sí, no es lo habitual pero es posible y se da en algunos casos. El software libre se caracteriza porque quien lo recibe tiene las *cuatro libertades*:

1. Usarlo para cualquier fin
2. Analizarlo y modificarlo
3. Distribuirlo tal y como está
4. Distribuir las modificaciones

Para esto, es necesario que quien lo reciba disponga del código fuente. Quien recibe el software puede venderlo, pero no puede evitar que el comprador pueda volver a revenderlo.

Ejercicio 2 (2.5 puntos)

¿Cuál es la diferencia entre `kill <pid>` y `kill -9 <pid>` ?
¿Cuándo debe usarse uno y cuándo debe usarse el otro?

Respuesta

`kill <pid>`

Envía al proceso `<pid>` la señal por omisión, SIGTERM, que solicita al proceso que concluya. Normalmente el proceso aceptará la orden y concluirá de forma ordenada, aunque también puede ignorar la orden, por haber sido programado así o por haberse colgado.

`kill -9 <pid>`

Envía al proceso la señal SIGKILL. En caso de que el emisor sea el dueño del proceso o el usuario root, esto causa que el proceso concluye de manera brusca, e inevitable.

Lo correcto es en primer lugar intentar finalizar un proceso con SIGTERM, y si falla, forzarlo con SIGKILL.

Ejercicio 3 (2.5 puntos)

Explica brevemente la diferencia entre trabajar sobre un VPS, *virtual private server* y trabajar sobre un PaaS, *platform as a service*.

(Como seguramente sabrás, ejemplos de PaaS son Heroku, Google App Engine y Microsoft Azure Web Sites. Un ejemplo de VPS es una instancia de EC2 de Amazon, esto es, una máquina en la *nube de amazon*)

Respuesta

Un VPS es una máquina virtual, completa, que un proveedor de *hosting* alquila a un usuario. Se administra prácticamente igual que una máquina física, con la única diferencia de que el hardware está virtualizado y controlado por el proveedor. En esta máquina, se puede ejecutar cualquier combinación de servicios.

Un PaaS es un tipo de computación en la nube. Un proveedor ofrece a sus clientes una plataforma para usar un servicio concreto, como pueda ser un servidor web, una base de datos, almacenamiento, etc. La administración de la máquina es responsabilidad del proveedor, el usuario solo usa y configura el servicio que haya contratado.

Ejercicio 4 (2.5 puntos)

Explica con tus palabras las diferencias y similitudes entre *unicode* y *UTF-8*.

Respuesta

Unicode es una norma definida en 1991 que enumera la práctica totalidad de los caracteres de los lenguajes humanos, naturales o artificiales, además de otros símbolos como los emoticonos. Informalmente podríamos decir que es un conjunto de tablas que asocia a cada símbolo un número.

UTF-8 es una de las formas posibles de codificar ese número. Detalla qué secuencia de bytes en concreto deben emplearse para indicar un símbolo unicode. Es relativamente reciente, del año 2006, pero se ha convertido en la codificación UTF-8 recomendada en

Unix y la más popular en internet. Microsoft Windows no usa UTF-8 de forma nativa pero muchas herramientas sí lo soportan.

Cualquier símbolo unicode puede codificarse en UTF-8, si bien es posible que en una plataforma concreta no pueda representarse gráficamente de forma correcta, porque no esté disponible la fuente tipográfica adecuada.

Laboratorio de administración y gestión de redes y sistemas
Prueba escrita - 15 de enero de 2017

Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Instrucciones:

Crea el directorio

`~/lagrs.enero.17`

y dentro de él, el fichero

`~/lagrs.enero.17/respuestas.TULOGIN.txt`

donde contestarás las preguntas. (La cadena TULOGIN representa tu nombre de usuario en el laboratorio, es decir, si tu login es pepito, el fichero se llamará ejercicio.pepito.txt).

El enunciado te pedirá que hagas una serie de cosas en tu ordenador, es conveniente que lo hagas realmente, aunque en realidad lo único que importa es lo que escribas en el examen, describiendo qué has hecho. Dicho de otro modo: el ordenador te servirá para comprobar que lo estás haciendo bien, pero es irrelevante lo que hagas en la shell, lo único que importa es lo que escribas en el examen.

Ejercicio 1 (1 punto)

Indica todos los pasos necesarios para montar por sshfs el directorio

`/var/tmp/`

de la máquina del laboratorio cuyo número es uno más que el tuyo, en el directorio

`/var/tmp/remota`

de tu máquina.

Ejemplo: si tu máquina es `kappa07`, monta el directorio `/var/tmp/` de `kappa08` en el directorio `/var/tmp/remota` de `kappa07`

Recuerda que si haces un paso, pero no cuentas en el examen que lo has hecho, es como si no lo hubieras hecho. Pega el resultado de la ejecución, de forma similar a la memoria de prácticas.

Ejercicio 2 (1.5 puntos)

Edita un fichero de texto llamado

`~/lagrs.enero.17/origenes.txt`

En su interior, escribe un texto que contenga en cada línea un nombre de fichero, incluyendo su path completo. Por ejemplo esto:

```
/tmp/1
/tmp/2
/tmp/3
```

1. Empleando las órdenes `xargs` y `touch`, haz que si no existen los ficheros indicados en el fichero de texto `origenes.txt`, se creen. (Si ya existen no importa lo que suceda con ellos)
2. Empleando las órdenes `xargs` y `touch`, haz que en el directorio de trabajo actual se cree un enlace simbólico a cada uno de los ficheros indicados en `origenes.txt`

Ejercicio 3 (2.5 puntos)

Crea un directorio llamado

```
~/lagrs.enero.17/base
```

Crea los ficheros

```
~/lagrs.enero.17/base/f1.txt
~/lagrs.enero.17/base/f2.txt
```

de forma que contengan un texto de ejemplo cualquiera, como *este es el fichero f1*

1. Haz todo lo necesario para que `/tmp/r1` y `/tmp/r2` y sean repositorios git cuyo contenido sea el mismo que base
2. Añade un fichero llamado f3.txt al repositorio r1
3. Propaga este cambio a r2

Observaciones:

Es necesario que respetes las convenciones habituales en git

Recuerda que no importa lo que hagas en el ordenador del laboratorio, sino lo que escribas en el examen

Ejercicio 4 (2 puntos)

1. Sea TUHOST la máquina del laboratorio que estás usando. Sea OTROHOST una máquina de laboratorio que esté vacía, funcionando y justo a tu derecha. (si no es posible, dile al profesor que te indique cuál es OTROHOST). Indica el nombre de TUHOST y OTROHOST, para que

conste en el examen
2. Haz todo lo necesario para que cuando lances romanclient.py contra el puerto 7777 de TUHOST, responda el romanserver.py del puerto 9999 de OTROHOST

Ejercicio 5 (1 puntos)

1. Indica qué es necesario para que tu script aguad de la práctica 5.3 se ejecute los lunes de septiembre, octubre y noviembre, cada 5 minutos
2. Indica qué es necesario para que ese mismo script se ejecute a las 9 de la mañana, de lunes a viernes

Ejercicio 6 (2 puntos)

Usando screen

1. Entra por ssh en otra máquina de este laboratorio
2. Lanza la orden top y déjala corriendo
3. Cierra la conexión con la máquina remota, sin que se interrumpa el top
4. Vuelve a entrar en la máquina remota y recupera la sesión

Recuerda que no importa lo que hagas en el laboratorio, solo lo que escribas en el examen.

Laboratorio de administración y gestión de redes y sistemas
Prueba escrita sobre las prácticas. 20 de diciembre de 2017.
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Instrucciones:

- Ejecuta `~mortuno/prepara_examen_lagrs` y comprueba que esto ha creado los siguientes ficheros

```
\verb|~/lagrs.diciembre.17/practico.jperez.txt|
\verb|~/lagrs.diciembre.17/contenedores_de.py|
\verb|~/lagrs.diciembre.17/cex/construye.sh|
\verb|~/lagrs.diciembre.17/cex/lanza_jpercex01.sh|
\verb|~/lagrs.diciembre.17/cex/context/Dockerfile|
\verb|~/lagrs.diciembre.17/cex/context/entrypoint.sh|
\verb|~/lagrs.diciembre.17/cex/context/holamundo|
```

- Como siempre, la cadena *jperez* representa tu nombre de usuario en el laboratorio y *jper*, las primeras 4 letras de tu nombre de usuario.
- Todos los ficheros estarán vacíos, excepto el script python, que contiene una función auxiliar.
- El enunciado te pedirá que hagas una serie de cosas en tu ordenador, es conveniente que lo hagas realmente, aunque lo único que importa es lo que escribas en los diferentes ficheros que indica el enunciado. Dicho de otro modo: el ordenador te servirá para comprobar que lo estás haciendo bien, pero **es irrelevante lo que hagas en la shell, lo único que importa es lo que escribas en los ficheros del examen**.

Ejercicio 1 (3 puntos)

En tu cuenta encontrarás un fragmento de script, con el siguiente nombre
`~/lagrs.diciembre.17/contenedores_de.py`
Completa el script, de forma que:

- Reciba un argumento por linea de comandos, que será un nombre de usuario. Basándose en la orden `docker images`, mostrará todas las imágenes de docker disponibles en el sistema, cuyo prefijo sea esa nombre de usuario. También sumará el total del tamaño de esas imágenes.

  Ejemplo:

```
mgarcia@alpha:~/lagrs.diciembre.17$ ./contenedores_de.py jperez

Contenedores con el prefijo jperez:
caa, cbb, cac
Tamaño total: 694 MB
```

Observaciones

- El nombre de usuario que se debe mostrar es el que se pasa por linea de comandos, no tu nombre de usuario ni el nombre del usuario que invoca el script.
- En el listado de imágenes que mostrará tu script, se omite el prefijo con el nombre de usuario. El nombre de usuario solo aparece 1 vez, al principio del informe.

  Dicho de otro modo: en el listado no debe decir *jperez/caa, jperez/cbb* sino *caa, cbb*. Tal y como se ve en el ejemplo.
- Para obtener los argumentos de línea de comandos, puedes usar `sys.argv[]`, no hace falta que uses la librería `optparse`.
- El script deber recibir un argumento, exactamente uno. En otro caso, el script debe mostrar un mensaje de error y finalizar sin hacer nada más. Si quieres, puedes levantar una excepción.
- En la salida de `docker images` (que es la entrada de tu script), el tamaño de cada imagen podrá estar expresado en kB, MB o GB. Pero la salidad de tu script siempre tendrá que estar expresada en MB. Para facilitarte esta parte, dentro del script ya tienes programada una función auxiliar, `any_size_to_mb()`, que recibe el tamaño expresado en cualquiera de las tres unidades y devuelve el tamaño expresado en MB. En el fuente del script tienes los detalles.
- En caso de que no haya ninguna imagen con ese nombre de usuario como prefijo, tu programa puede mostrar lo que quieras: un mensaje especial, una salida vacía, un tamaño de 0, un tamaño en blanco, etc. Lo único que no es admisible es que el programa falle, muestre algo sin sentido o una excepción.
- Cuando lo tengas funcionando, enséñaselo al profesor.

Ejercicio 2 (4 puntos)

En este ejercicio prepararás una imagen de un contenedor según la siguiente especificación:

1. Una vez lanzando, tendrá corriendo un servidor de ssh.
2. Tendrá creado un usuario llamado *alumno*.

3. Ese usuario tendrá asignada la contraseña *xx99*. Esto no lo has hecho en ninguna práctica, pero la transparencia 17 del tema de administración de usuarios indica cómo cambiar una contraseña desde un script.

4. Cuando el alumno entre en el contenedor, podrá ejecutar un script llamado

   `/usr/local/bin/holamundo`

   que mostrará por la salida estándar un mensaje similar a `"¡Hola, mundo!"`

5. La imagen se llamará `jpercex`, donde `jper` representa las primeras 4 letras de tu nombre de usuario en el laboratorio.

6. El contenedor se llamará `jpercex01`

7. Para preparar este contenedor, edita los ficheros

   `~/lagrs.diciembre.17/cex/construye.sh`

   `~/lagrs.diciembre.17/cex/lanza_jpercex01.sh`

   `~/lagrs.diciembre.17/cex/context/Dockerfile`

   `~/lagrs.diciembre.17/cex/context/entrypoint.sh`

   `~/lagrs.diciembre.17/cex/context/holamundo`

   Como es habitual, *jper* representa las primeras 4 letras de tu nombre de usuario en el laboratorio.

8. Se valorará positivamente que instales solo los paquetes necesarios y que uses solo los ficheros necesarios. Esto es, si el ejercicio funciona, tendrás buena nota. Pero para tener la máxima nota tendrás que instalar solo los ficheros y paquetes imprescindibles (y no otros que hayan sido necesarios para otros ejercicios)

9. Cuando lo tengas funcionando, enséñaselo al profesor.

Sugerencia: para probar el servidor de ssh, entra por ssh desde el propio contenedor hasta el contenedor, usando la dirección *localhost*.

Ejercicio 3 (3 puntos)

Asegúrate de que estás sentado en un puesto con número de máquina superior a 5. Por tanto, no admite conexiones del exterior. Haz todo lo necesario para:

1. Lanzar un servidor de vnc en ese puesto. No elijas puerto, usa el que asigne el servidor. Pon el tamaño del escritorio y la profundidad de color que prefieras.

2. Configurar un túnel inverso de SSH, usando el mismo proxy y el mismo puerto que en tu práctica de túnel inverso (no el puerto de tu práctica de vnc en dockerserver), para que un cliente de vnc (vinagre), pueda acceder a tu servidor de vnc, a través de la dirección pública del proxy.

3. Ejecutar el cliente vnc, en el puesto de tu derecha, de forma que se conecte al servidor de vnc.

Dibuja en papel un diagrama de tu configuración. Escribe (en el fichero `practico.TULOGIN.txt`) los pasos que has seguido y cuando lo tengas funcionando, enséñaselo todo al profesor.

Laboratorio de Administración y Gestión de Redes y Sistemas
Prueba escrita sobre la teoría. 20 de diciembre de 2017
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Instrucciones:

- Encontrarás en el *home* del puesto del laboratorio el fichero el fichero `~/teoria.tulogin.txt`.

  Cámbiale el nombre, reemplazando *tulogin* por tu verdadero login. Por ejemplo, si eres *jperez*, debes hacer

  `mv teoria.tulogin.txt teoria.jperez.txt`

  Revisa que esté bien hecho. Si te equivocas en este paso tan sencillo, suspenderás.

- Dentro del fichero `teoria.jperez.txt`, escribe tu login, nombre y apellidos y contesta al examen.

- Recuerda que si te limitas a repetir la información contenida en las transparencias, tu respuesta no tendrá ningún valor

Ejercicio 1

¿Que es VT-x? ¿En qué consiste?

Respuesta

Es una tecnología presente en los procesadores Intel desde hace unos 10 años, que permite la virtualización nativa. Varios núcleos de varios sistemas operativos pueden correr sobre el mismo procesador, de forma que cada uno de ellos percibe estar en el *ring 0*, esto es, cada uno percibe un acceso exclusivo e ilimitado a la CPU, sin acceso a los recursos de otros núcleos.

En tecnologías anteriores es necesaria una capa de software adicional para distribuir la CPU entre los núcleos de cada sistema operativo.

Ejercicio 2

Supongamos que estoy en una máquina Unix/Linux/MacOS y hago lo siguiente:

- Copio un script `holamundo.py` en mi directorio `~/bin`.
- Le doy permisos de ejecución para mi usuario.
- Escribo en la terminal `holamundo.py` pero no se ejecuta.
- Me doy cuenta de que mi varible no incluye `~/bin` así que escribo en el terminal `PATH=$PATH:$HOME/bin`

- Vuelvo a escribir `holamundo.py`

1. ¿Qué sucede? ¿Por qué?
2. Ahora abro un terminal nuevo, en otra ventana. ¿Qué sucede? ¿Por qué?

Observaciones

- Contesta solamente las preguntas 1 y 2. Los puntos anteriores son el contexto del ejercicio, no se te pide que escribas esas órdenes.

Respuesta

1. Hemos añadido a la variable de entorno PATH el directorio donde está el fichero. La shell buscará el fichero en todos los directorios indicados en PATH, por tanto ahora lo encontrará y ejecutará.
2. En un terminal nuevo la shell será un proceso distinto, con sus propias variables de entorno. En esta shell, no hemos cambiado la variable PATH, así que la shell no encontrará el script y no lo ejecutará.

   Para ello deberíamos haber añadido la sentencia `export PATH=$PATH:$HOME/bin` al fichero que ejecuta cada shell antes de iniciarse. Lo más recomendable es el fichero `~/.bashrc`, teniendo en cuenta que el fichero `.bash_profile` debería llamar a `~/.bashrc`, de lo contrario, el `~/.bashrc` no será invocado en las shell interactivas no de login, como por ejemplo un acceso por ssh.

Ejercicio 3

¿Cuál es la diferencia entre los siguientes directorios, cuál es su propósito?
`/lib`
`/usr/lib`
`/usr/local/lib`

Respuesta

- `/lib`

  Contiene librerías esenciales para ejecutables del sistema.

- `/usr/lib`

  Contiene librerías para ejecutables de menor importancia, no esenciales.

- `/usr/local/lib`

  Contiene librerías de programas no incluidos de forma estándar en la distribución, sino que cada administrador local añade.

Ejercicio 4

Reproducimos aquí una oferta real de empleo vista recientemente. A partir de tus conocimientos sobre el significado del término *DevOps*, coméntala. Si tienes alguna duda con el inglés puedes consultar al profesor.

- DEVOPS ENGINEERS

  Your activities will mainly focus on the design and development of a software application and/or embedded systems. It is crucial that you have a good overview of possible software design methodologies, operating systems and programming languages and that you are able to motivate this to others.

- WE ARE LOOKING FOR

  You have a B. Sc. or M.Sc. degree in Computer Sciences, Informatics or similar education with several years of experience in software. During your M.Sc. or through professional experience you have gained expertise in one or more of the following topics:

  · Work experience as System Engineer supporting web application architectures (Java, Tomcat, Apache, JBoss, Drools, etc)

  · Experience working in Agile projects applying methodologies (Jira, Confluence, BitBucket, Bamboo etc)

  · Continuous Integration tools (Maven, Jenkins, etc.)

  · Experience with automation/configuration management using Puppet and/or other tools

  · Virtualization technologies (VMware, Docker, AWS)

  · Strong scripting skills in order to automate common tasks (bash, Unix tools)

  · Knowledge of best practices and IT operations in an always-up, always-available service

  · Experience developing logical model for BRMS

  · Fluency in English and Spanish

Respuesta

Según todos los especialistas en *DevOps*, los libros, los congresos, etc, no tiene sentido hablar de *Ingeniero DevOps*. Por ejemplo en el libro *DevOps for developers* (Michael Huttermann, Ed, Apress) podemos leer explícitamente *DevOps is not a new job. If you see a job advertisement that asks for a DevOps expert, please point the author of the ad to this book.*

Pero en la industria informática general, entre los no especialistas en *DevOps* es relativamente frecuente usar el término de forma distinta, como vemos por ejemplo en este anuncio. Se solicita un desarrollador con conocimientos en metodologías ágiles, integración continua, administración con herramientas de configuración de sistemas, etc, esto, es, herramientas que son habituales en *DevOps*. Pero ni son imprescindibles esas herramientas para hacer *DevOps* ni su uso implica estar siguiendo metodologías *DevOps*.

Laboratorio de administración y gestión de redes y sistemas
Examen de prácticas. 14 de diciembre de 2018.
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Instrucciones

- Abre una sesión en el laboratorio con tu usuario y contraseña habituales.
- Ejecuta el script `prepara_lagrs`
  Esto creará en tu cuenta el directorio `~/lagrs.diciembre.18`
  y los ficheros `~/lagrs.diciembre.18/practico.tulogin.txt`,
  `~/lagrs.diciembre.18/claves.tgz` y
  `~/lagrs.diciembre.18/vigila_puertos.py`

Ejercicio 1. (2 puntos)

En el fichero `~/lagrs.diciembre.18/claves.tgz`
tienes todo lo necesario para abrir una sesión ssh en la máquina 193.147.79.2 con el usuario *alumno*. También ficheros que *sobran*. Entra en esa máquina, escribe en `practico.tulogin.txt` los pasos que has seguido y muestra la sesión al profesor.

Ejercicio 2. (4 puntos)

En este ejercicio escribirás un script en python que, usando la order netstat, vigilará ciertos puertos y enviará un mensaje de telegram si algún proceso escucha peticiones en ellos. Edita el fichero `~/lagrs.diciembre.18/vigila_puertos.py` para que se corresponda con la siguiente especificación:

1. El programa leerá de un fichero de texto con nombre `id_usuario.txt` el identificador de usuario de telegram que recibirá las alarmas. No especifiques ningun *path*, de forma que se busque en el directorio actual.
2. El programa leerá el token del bot de telegram en el fichero `token.txt`. De nuevo, no especifiques ningún trayecto, que el script lea el fichero desde el directorio actual.
3. En las primeras líneas del script define una lista parecida a esta:

   ```
   PUERTOS_TCP = [6666, 443]
   ```

4. Cada vez que se ejecute el programa, revisará todos los puertos TCP especificados en PUERTOS_TCP, y si alguno de ellos está ocupado, en cualquier estado (ya sea *ESCUCHAR*, *ESTABLECIDO*, o cualquier otro en cualquier otro idioma: *LISTEN*, *ESTABLISHED*...), enviará una alarma a través de telegram describiendo toda la información relevante proporcionada por netstat.
5. La alarma solo se disparará si el puerto está ocupado en alguna dirección local, sea cual sea, pero no en ninguna dirección remota.
6. Usa romanserver.py para probar el programa. Cuando funcione, enséñaselo al profesor.

Ejercicio 3. (4 puntos)

En este ejercicio prepararás y lanzarás dos contenedores Docker, uno con *romanserver.py* y otro con *romanclient.py*, según la siguiente especificación:

1. Ambos contenedores estarán conectados por un segmento privado de red dentro de Docker.
2. Como todos los contenedores del laboratorio comparten segmento de red virtual, cada alumno necesita su propio puerto. El profesor te indicará un *puerto_alumno*, usa el puerto TCP 12000 + *puerto_alumno*.
3. El contenedor con romanserver.py estará basado en una imagen llamada `exa`. Si tu nombre de usuario fuera *jperez*, el contenedor se llamaría `jperexa01`. Sustituye *jper* por los primeros 4 caracteres de tu nombre de usuario.

   Al ejecutarlo, lanzará el programa `romanserver.py` en el puerto reservado para tí y a continuación abrirá una shell.
4. El contenedor con romanclient.py estará basado en una imagen llamada `exb`.

   Al ejecutarlo, lanzará un terminal, donde el usuario podrá invocar a romanclient.py con los parámetros adecuados. El usuario deberá podar lanzar romanclient.py sin especificar ningún path.
5. El contenedor exa también también tendrá disponible romanclient.py, para poder probar el servicio en local.
6. Ambos contenedores deberán poderse hacer *ping* entre sí. También podrán usar *ifconfig*.
7. Usa el mismo convenio empleado en clase para nombrar los ficheros de configuración de los contenedores, creación de imágenes y ejecución. Aunque guardando todo en el directorio `~/lagrs.diciembre.18`. Esto es, usa los siguientes ficheros:

   ```
   ~/lagrs.diciembre.18/exa/construye.sh
   ~/lagrs.diciembre.18/exa/lanza_jperexa01.sh
   ~/lagrs.diciembre.18/exa/context/Dockerfile
   ~/lagrs.diciembre.18/exa/context/entrypoint.sh
   ~/lagrs.diciembre.18/exa/context/romanserver.py
   ~/lagrs.diciembre.18/exa/context/romanclient.py
   ```

   Y para el cliente:

   ```
   ~/lagrs.diciembre.18/exb/construye.sh
   ~/lagrs.diciembre.18/exb/lanza_jperexb01.sh
   ~/lagrs.diciembre.18/exb/context/Dockerfile
   ~/lagrs.diciembre.18/exb/context/entrypoint.sh
   ~/lagrs.diciembre.18/exb/context/romanclient.py
   ```
8. Dentro de los contenedores, los ficheros romanserver.py y romanclient.py deben estar en el sitio *correcto* según la norma FHS.
9. Instala solo los paquetes necesarios, usa solo los ficheros necesarios (y no otros que hayan hecho falta en las prácticas).
10. Cuando lo tengas funcionando, enséñaselo al profesor.

Laboratorio de Administración y Gestión de Redes y Sistemas
Examen de teoría. 14 de diciembre de 2018
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Instrucciones:

- Entra en el puesto del laboratorio con el nombre de usuario *examen* y la contraseña *examen*.
- Ejecuta el script `prepara_lagrs`
- Esto dejará en el ordenador el fichero `~/lagrs/TULOGIN/teoria.TULOGIN.18.txt`, donde debes escribir tus respuestas. TULOGIN será tu nombre de usuario en el laboratorio.

Ejercicio 1

Explica brevemente qué hacen las siguientes órdenes:

1. `su`
2. `su xxx`
3. `sudo xxx`
4. `sudo su`

Respuesta

1. Solicita la contraseña de root. Si es correcta, la shell actual (la sesión) pasa a ser de root.
2. Solicita la constraseña del usuario xxx. Si es correcta, la shell actual (la sesión) pasa a ser de ese usuario. Si esta orden se invoca con privilegios de root, no solita la contraseña del usuario xxx.
3. Solicita al usuario su contraseña. Si es correcta y el usuario está autorizado en el fichero *sudoers*, la orden xxx se ejecutará con privilegios de root. Si el usuario introdujo su contraseña recientemente, no se le vuelve a pedir.
4. Igual que el paso anterior, pero la orden que se ejecuta es *su*, por tanto, la shell pasa a ser de root.

Ejercicio 2

En la metodología *scrum*, explica brevemente la diferencia entre un *sprint* y una *historia de usuario*.

Respuesta

Un *sprint* es el periodo de tiempo en el que se desarrola una nueva versión del producto software. Su duración suele ser de entre 2 y 4 semanas.

Una *historia de usuario* es una descripción informal en lenguaje natural de una nueva característica en el software, visto desde el punto de vista del usuario.

Ejercicio 3

Si editamos algo en el fichero `.bash_profile` o en el fichero `.bashrc`, los efectos no son inmediatos. La solución del principiante es *salir y entrar*. Esto funciona, pero hay una forma mejor.

1. ¿Por qué funciona *salir y entrar*?
2. ¿Cuál es la solución mejor?
3. ¿Por qué funciona la *solución mejor*?

Respuesta

1. Porque si cerramos la sesión y la volvemos a abrir, se leen de nuevo los ficheros de invocación de la shell (*.bashrc*, *.bash_profile*, etc). Por supuesto, si apagamos el ordenador y encendemos de nuevo, también.
2. Ejecutar explícitamente los ficheros de invocación de la shell con la orden *source*.
3. Porque ejecuta estos ficheros en el contexto de la misma shell.

Ejercicio 4

Explica brevemente las diferencias entre una instalación basada en clonación y una instalación basada en los ficheros *preseed* de Debian y derivados.

Respuesta

- Clonación

  El administrador configura una máquina. El disco duro de esa máquina se copia en las demás. Esto exige que el hardware sea idéntico

- Ficheros *preseed*

  El sistema operativo se instala desde cero en cada máquina. Este fichero contiene las respuestas a las preguntas que se le hacen al administrador en las instalaciones manuales ordinarias. Tiene la ventaja de que la instalación se adapta a cada hardware particular.

Laboratorio de administración y gestión de redes y sistemas
Examen práctico. 14 de junio de 2019.
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Preparativos:

- Crea un directorio `~/lagrs.junio.19/`, donde escribirás todo lo indicado a continuación.

Ejercicio 1 (5 puntos)

En tu práctica 3.5 escribirte un programa que revisaba todos los directorios indicados en la constante `DIRECTORIOS` para comprobar que ni el tamaño ni el número de ficheros contenidos excediera un límite prefijado. Si se excedía algún límite, se enviaba un mensaje de alarma mediante telegram. Ahora modificarás ese programa para que además de hacer lo que ya hacía, compruebe que los directorios y su contenido pertenezca a ciertos usuarios previstos.

1. Copia el script de tu práctica 3.5 en `~/lagrs.junio.19/monitor.py`.
2. Añade al principio del script una línea similar a esta:

   ```
   USUARIOS_PERMITIDOS = ["jperez", "mgarcia"]
   ```

3. Tu programa tiene que comprobar que todos los directorios indicados en la constante `DIRECTORIOS` pertenezcan a alguno de estos usuarios permitidos. Si alguno de estos directorios pertenece a otro usuario, enviará un mensaje de telegram.
4. Del mismo modo, tu programa también tiene que avisar si alguno de los ficheros o directorios contenidos dentro de estos directorios pertenecen a un usuario que no está entre los permitidos.
5. No es necesario que tu programa recorra recursivamente los subdirectorios de `DIRECTORIOS` revisando los dueños de ficheros y directorios.
6. Si el script se lanza con la opción `-f` o `--force-telegram`, siempre enviará un mensaje de telegram, ya sea para indicar que hay problemas o para indicar que todo está bien (tamaño de los directorios, número de ficheros y dueños de los directorios y ficheros).
7. Enséñaselo al profesor cuando acabes.

Ejercicio 2 (5 puntos)

En este ejercicio prepararás un contenedor docker donde el usuario *root* tendrá el editor vim listo para usar, con un fichero de configuración de vim personalizado. Para ello, solo tienes que instalar vim dentro del contedor y copiar en el directorio personal del usuario *root* (esto es, en el directorio `/root` del contenedor) un fichero oculto con nombre `.vimrc` y con el siguiente contenido

```
syntax on
set tabstop=4
set nu
```

El contenedor estará construido a partir de ubuntu 18.04, con los paquetes actualizados a la última versión disponible. Estará configurado en español y tendrá vim instalado. (Nada más, si añades paquetes o configuración adicional, tendrás una penalización en la nota). Los ficheros necesarios para esto seguirán el mismo convenio que hemos venido usando durante toda la asignatura.

1. El contenedor basado en una imagen llamada `exa`. Si tu nombre de usuario fuera *jperez*, el contenedor se llamaría `jperexa01`. Sustituye *jper* por los primeros 4 caracteres de tu nombre de usuario.

2. Al ejecutar el contenedor, se abrirá automáticamente una sesión de vim. Y después, cuando se cierre este vim, se abrirá automáticamente una sesión de shell bash.

3. Los nombres de los ficheros que debes preparar para esto serán:

   ```
   ~/lagrs.junio.19/exa/construye.sh
   ~/lagrs.junio.19/exa/lanza_jperexa01.sh
   ~/lagrs.junio.19/exa/context/Dockerfile
   ~/lagrs.junio.19/exa/context/entrypoint.sh
   ~/lagrs.junio.19/exa/context/vimrc
   ```

   (Reemplazando *jper* por las primeras 4 letras de tu login)

4. Observa que el fichero de configuración de vim es un fichero oculto, empieza por punto. Pero cuando lo prepares en el directorio contexto, no debe estar oculto (su nombre no empieza por punto). En otras palabras: tienes que preparar un fichero llamado `~/lagrs.junio.19/vimrc`, que se copie dentro de la imagen del contenedor con nombre `/root/.vimrc`

5. Enséñaselo al profesor cuando acabes.

Laboratorio de Administración y Gestión de Redes y Sistemas
Examen de teoría. 14 de junio de 2019
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Preparativos

- Ejecuta el script `prepara`
- Esto creará en tu ordenador el fichero `~/lagrs/TULOGIN/teoria.txt`, donde TULOGIN es tu nombre de usuario en el laboratorio. Escribe tus respuestas en este fichero.

Ejercicio 1 (2.5 puntos)

La orden *sudo*. ¿Para qué sirve? ¿Cómo se usa? ¿Cómo se configura?

Ejercicio 2 (2.5 puntos)

Explica las diferencias entre las máquinas virtuales *de proceso* y las máquinas virtuales *de sistema*. Pon algún ejemplo de cada una.

Ejercicio 3 (2.5 puntos)

Un túnel de ssh remoto, también llamado túnel inverso, sirve para acceder a un servidor tras un NAT. Explica, a nivel conceptual, qué hace, por qué es útil, cuáles serían técnicas alternativas para acceder al servidor. No es especialmente importante que indiques los parámetros concretos para configurar el túnel.

Ejercicio 4 (2.5 puntos)

Explica brevemente las diferencias fundamentales entre el desarrollo de software *en cascada* y el desarrollo de software *ágil*.

Laboratorio de administración y gestión de redes y sistemas
Examen práctico. 13 de enero de 2020.
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Preparativos:

- Crea un directorio `~/lagrs.enero.20/`, donde escribirás todo lo indicado a continuación.

Ejercicio 1 (5 puntos)

En este ejercicio escribirás un script en python 3 que, usando la order netstat, vigilará ciertos puertos y enviará un mensaje de telegram si algún proceso escucha peticiones en ellos. Edita el fichero `~/lagrs.enero.20/vigila_puertos.py` para que se corresponda con la siguiente especificación:

1. El programa leerá de un fichero de texto con nombre `id_usuario.txt` el identificador de usuario de telegram que recibirá las alarmas. No especifiques ningun *path*, de forma que se busque en el directorio actual.

2. El programa leerá el token del bot de telegram en el fichero `token.txt`. De nuevo, no especifiques ningún trayecto, que el script lea el fichero desde el directorio actual.

3. En las primeras líneas del script define una lista parecida a esta:

   ```
   PUERTOS_TCP = [6666, 443]
   ```

4. Cada vez que se ejecute el programa, revisará todos los puertos TCP especificados en PUERTOS_TCP, y si alguno de ellos está ocupado, en cualquier estado (ya sea *ESCUCHAR*, *ESTABLECIDO*, o cualquier otro en cualquier otro idioma: *LISTEN*, *ESTABLISHED*...), enviará una alarma a través de telegram describiendo toda la información relevante proporcionada por netstat.

5. La alarma solo se disparará si el puerto está ocupado en alguna dirección local, es irrelevante lo que pase en direcciones remotas.

6. Usa romanserver.py para probar el programa. Cuando funcione, enséñaselo al profesor.

Ejercicio 2 (5 puntos)

En este ejercicio prepararás un contenedor docker que al ejecutarse:

1. El usuario root tendrá una sesión de shell.
2. Este usuario tendrá un directorio `~/bin`.
3. Tendrá en `~/bin/tictac` el siguiente script

```
#!/bin/bash
while true
do
    sleep 1
    echo -n "tic"
    sleep 1
    echo " tac"
done
```

4. La variable de entorno PATH del usuario root será la que prepare normalmente la distribución, pero añadiendo el directorio *bin* del *home* del usuario root. De tal forma que se podrá lanzar el script anterior tecleando simplemente `tictac`, desde cualquier directorio, sin añadir ningún *path*.

El contenedor estará construido a partir de ubuntu 18.04. No actualices los paquetes y déjalo en inglés. (Si añades paquetes o configuración adicional, tendrás una penalización en la nota). Los ficheros necesarios para esto seguirán el mismo convenio que hemos venido usando durante toda la asignatura.

1. El contenedor estará basado en una imagen llamada `exa`. Si tu nombre de usuario fuera *jperez*, el contenedor se llamaría `jperexa01`. Sustituye *jper* por los primeros 4 caracteres de tu nombre de usuario.
2. Los nombres de los ficheros que debes preparar para esto serán:

```
~/lagrs.enero.20/exa/construye.sh
~/lagrs.enero.20/exa/lanza_jperexa01.sh
~/lagrs.enero.20/exa/context/Dockerfile
~/lagrs.enero.20/exa/context/entrypoint.sh
~/lagrs.enero.20/exa/context/bashrc
~/lagrs.enero.20/exa/context/tictac
```

   (Reemplazando *jper* por las primeras 4 letras de tu login)

3. Observa que el fichero de configuración `.bashrc` es un fichero oculto, empieza por punto. Pero cuando lo prepares en el directorio contexto, no debe estar oculto (su nombre no empieza por punto). En otras palabras: tienes que preparar un fichero llamado `bashrc`, que dentro del contenedor tendrá por nombre `.bashrc`
4. Enséñaselo al profesor cuando acabes.

Laboratorio de Administración y Gestión de Redes y Sistemas
Examen de teoría. 13 de enero de 2020
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Preparativos

- Ejecuta el script `prepara`
- Esto creará en tu ordenador el fichero `~/lagrs/TULOGIN/teoria.txt`, donde TULOGIN es tu nombre de usuario en el laboratorio. Escribe tus respuestas en este fichero.

Ejercicio 1 (2.5 puntos)

1. Un usuario de Unix ¿puede pertenecer a más de un grupo? En caso afirmativo ¿Es habitual? ¿Qué sentido tiene esto?
2. Explica brevemente que es el *grupo primario*.
3. Explica brevemente que es el *grupo primario por omisión*.

Ejercicio 2 (2.5 puntos)

Explica con detalle esta sesión de Unix: qué es cada linea, cada órden, qué hace, qué sucede y por qué.

```
01 koji@mazinger:~$ echo hola > fichero.txt
02 koji@mazinger:~$ file fichero.txt
03 fichero.txt: ASCII text
04 koji@mazinger:~$ echo niño >> fichero.txt
05 koji@mazinger:~$ file fichero.txt
06 fichero.txt: UTF-8 Unicode text
```

(Evidentemente los números iniciales no forman parte de las órdenes, solo aparecen para facilitar la referencia)

Ejercicio 3 (2.5 puntos)

Continuous Integration, *Continuous Delivery* y *Continuous Deployment*:

1. Explica brevemente en qué consiste cada una de estas tres técnicas.
2. Cuando no existían ¿Cuál era la manera habitual de trabajar? ¿Qué problemas tenía?

Ejercicio 4 (2.5 puntos)

Para administrar un conjunto de máquinas Unix podemos usar dos tipos de herramientas:

- Scripts desarrollados por nosotros, en bash, en python o en algún lenguaje similar. Accediendo a las máquinas por ssh o métodos parecidos.
- Software de administración como *Ansible*, *Puppet* o *Chef*

Contesta brevemente

1. ¿Cuáles son las diferencias principales entre trabajar con técnicas del primer grupo o con herramientas del segundo grupo?
2. Si usamos herramientas del segundo grupo ¿Cuáles son las diferencias principales entre *Ansible* y *Puppet*?

Laboratorio de administración y gestión de redes y sistemas
Examen teorico-práctico parcial. 25 de noviembre de 2020.
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

- Entra en tu cuenta del laboratorio y ejecuta

  `~mortuno/prepara`

- Esto creará el fichero `~/parcial.2020.TULOGIN.txt` (donde TULOGIN es tu nombre de usuario). Contesta las dos preguntas en este fichero, solo aquí, en ningún otro sitio.

Ejercicio 1 (3 puntos)

Supongamos que tienes que tienes que diseñar el sistema informático basado en Linux para un colegio con unos pocos cientos de ordenadores. La solución podría basarse tanto en virtualización como en clonación. Explica brevemente en qué consiste cada enfoque, cuáles son sus ventajas y cuáles sus inconvenientes. Sin duda en tu respuesta habrá *dependes*. En esos casos, considera las opciones posibles. Ejemplo: Si los puestos son relativamente potentes entonces ..., en otro caso .... Si son muy heterogéneos, ..., y si no, ...

Ejercicio 2 (7 puntos)

Deseamos un par de contenedores con la siguiente especificación:

- Estarán basados en la misma imagen.
- Tendrán ambos el servicio *iperf* listo para atender peticiones TCP en el puerto 9999.
- Montarán ambos por **sshfs** en el directorio */media/(lab)* tu *home* de la maquina *f-l2108-pc02*. Supodremos que está funcionando bien, en un sistema más robusto habría que verificarlo, pero aquí lo omitimos). Observa que **no** se te pide un montaje *bind*.

  La autenticación será manual, esto es, al arrancarse el contenedor, te pedirá tu contraseña en el laboratorio.

  Esto no lo has hecho en prácticas (usar sshfs dentro de un contenedor). Pero sí lo vimos en clase de teoría. Recuerda que la orden `docker run` necesitará ciertos parámetros, documentados en las transparencias sobre Docker.

Especifica los ficheros necesarios para conseguir esto en Docker (una versión moderna, la incluida Ubuntu 20.04):

1. Indica qué ficheros habrá en el directorio *context* y con qué contenido.
2. Al igual que en prácticas, queremos un script para preparar la imagen. Indica qué nombre tendría y qué contenido.
3. Al igual que en prácticas, queremos un script para lanzar cada contenedor. Indica qué nombre tendrían y qué contenido.
4. Al lanzar los contenedores, el usuario escribirá su contraseña, a continuación deberá averiguar la IP del *otro* contenedor y lanzar el cliente de iperf. Indica cómo hacer estos pasos.

Solución

- Fichero `context/Dockerfile`

```
FROM ubuntu:20.04

RUN apt update && apt upgrade -y
RUN apt-get install -y net-tools iputils-ping sshfs iperf

COPY entrypoint.sh /

EXPOSE 9999
CMD ["/entrypoint.sh"]
```

- Fichero `context/entrypoint.sh`.

```
#!/bin/bash
iperf -s -p 9999 &
mkdir /media/lab
sshfs mortuno@f-l2108-pc02.aulas.etsit.urjc.es: /media/lab
/bin/bash
```

- Script para crear la imagen, `construye.sh`

```
#!/bin/bash
docker build -t jperez/exa context
```

- Script para lazar el primer contenedor, `lanza_jperexa01.sh`

```
#!/bin/bash
PREFIJO=jper
IMAGEN=exa
USUARIO=jperez
CONTENEDOR=$PREFIJO${IMAGEN}01
docker run --rm -it -h $CONTENEDOR --name $CONTENEDOR   \
    --cap-add SYS_ADMIN --device /dev/fuse  \
    --security-opt apparmor:unconfined \
    $USUARIO/$IMAGEN
```

  El script para lanzar el segundo contenedor, `lanza_jperexa02.sh` es idéntico, excepto la línea `CONTENEDOR=$PREFIJO${IMAGEN}02`

- Todos los scripts necesitan permiso de ejecución

```
\verb|chmod ugo+x context/entrypoint.sh|
\verb|chmod ugo+x *.sh|
```

- Una vez lanzados los contendores, introducimos la contraseña, averiguamos la dirección IP de cada uno con la orden *ifconfig* y ejecutamos

```
iperf -c <IP_SERVIDOR> -p 9999
```

  Donde `<IP_SERVIDOR>` es la dirección IP del contenedor donde está el servidor iperf.

Laboratorio de administración y gestión de redes y sistemas
Examen del primer parcial. 4 de febrero de 2021.
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

- Entra en tu cuenta del laboratorio y ejecuta

  `~mortuno/prepara`

- Esto creará el fichero `~/lagrs.feb.21/parcial.TULOGIN.txt` (donde TULOGIN es tu nombre de usuario). Contesta la pregunta en este fichero, solo aquí, en ningún otro sitio.

Ejercicio único

En este ejercicio indicarás todo lo necesario para preparar un contenedor docker según la siguiente especificación:

1. En el contenedor habrá un usuario con el mismo nombre que tu cuenta del laboratorio.
2. Al poner en marcha el contenedor, en el terminal habrá una sesión de shell de este usuario, que tendrá el multiplexor de terminales *tmux* listo para usar.
3. *tmux* estará personalizado de forma que la tecla *bind* no sea `Ctrl b` sino `Ctrl a`. Para ello, solo hay que instalar el paquete *tmux* dentro del contenedor y copiar en el directorio personal del usuario un fichero oculto con nombre `.tmux.conf` y con el siguiente contenido:

   ```
   set -g prefix C-a
   bind C-a send-prefix
   unbind C-b
   ```

4. El contenedor estará construido a partir de ubuntu 20.04, con los paquetes actualizados a la última versión disponible. Estará configurado en español y tendrá *tmux* instalado. (Nada más, si añades paquetes o configuración adicional, tendrás una penalización en la nota).
5. El contenedor estará basado en una imagen llamada `exa`. Si tu nombre de usuario fuera *jperez*, el contenedor se llamaría `jperexa01`. Sustituye *jper* por los primeros 4 caracteres de tu nombre de usuario.
6. Siguiendo el mismo convenio que hemos venido usando durante toda la asignatura, los nombres de los ficheros que habría que preparar serían.

   ```
   ~/lagrs.feb.21/exa/construye.sh
   ~/lagrs.feb.21/exa/lanza_jperexa01.sh
   ~/lagrs.feb.21/exa/context/Dockerfile
   ~/lagrs.feb.21/exa/context/entrypoint.sh
   ~/lagrs.feb.21/exa/context/tmux.conf
   ```

   (Reemplazando *jper* por las primeras 4 letras de tu login). Pero no escribas en esos ficheros, **Escribe solamente en parcial.TULOGIN.txt** indicando qué habría en cada fichero. P.e. *Contenido del fichero X: blabla. Contenido del fichero Y: blabla*
7. Observa que el fichero de configuración de *tmux* es un fichero oculto, empieza por punto. Pero cuando lo prepares en el directorio contexto, no debe estar oculto (su nombre no empieza por punto). En otras palabras: tienes que preparar un fichero llamado `~/lagrs.feb.21/tmux.conf`, que se copie dentro de la imagen del contenedor, en el *home* del usuario, con nombre `.tmux.conf`

Laboratorio de administración y gestión de redes y sistemas
Examen práctico (segundo parcial)
4 de febrero de 2021.

Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Ejercicio unico (10 puntos)

En tu cuenta encontrarás

- El fichero `~/lagrs.feb.21/ckcron.TULOGIN.py`, que debes completar para que sea un programa que revise una serie de tablas de cron. Además de detectar errores, dará avisos sobre aspectos de la tabla que, siendo correctos, pueden ser peligrosos.
- Una serie de ficheros de texto `~/lagrs.feb.21/tablaNN.txt` con tablas de cron, que también aparecen impresas a continuación. Estas tablas serán los casos de prueba para tu programa. Algunas son correctas, otras tienen errores o situaciones que consideramos peligrosas. Tu programa debe detectar todos los problemas que puedan tener estos ejemplos. Puedes ignorar los posibles errores no previstos aquí.

Tu programa:

- Recibirá una lista de argumentos por línea de comandos, que serán nombres de ficheros. Usa *sys.argv*, es suficiente para este caso. Si te sobra tiempo, puedes reemplazarlo por *optparse*.

  Ejemplo

  ```
  ckcron.jperez.py  tabla01.txt tabla02.txt tabla03.txt
  ```

- Para cada una de esas tablas, tu programa deberá decir si les ha detectado algún error o no. (El programa no debería decir que son *correctas* porque no hará una comprobación exhaustiva). En caso de error, deberá describirlo brevemente.
- El programa mostrará un *aviso* si algún comando de la tabla está indicado como un nombre de fichero, sin especificar *path*.
- El programa mostrará un *aviso* si algún comando (con *path* completo) no existe. Tal vez no sea un error porque tal vez sí exista cuando se ejecute, pero consideramos que esta situación merece al menos un aviso. (No importa si un comando sin trayecto existe o no, simplemente notifica el aviso por falta de trayecto)
- El programa mostrará un *aviso* si alguna variable de entorno no está definida. Tal vez no sea un error, tal vez se definirá en su momento, pero merece un aviso.
- Naturalmente, el programa no solo debe funcionar para esas tablas en particular, sino que debe admitir cualquier otra tabla con problemas semejantes.
- Deja claro en el código fuente qué tipo de error estás buscando. Ejemplo: *#Ahora detectamos si hay texto en klingon como en tabla12.txt, que es incorrecto.*

Observaciones

- Encontrarás en el esqueleto de fichero una función sencilla que, a partir de una línea de texto, devuelve una lista de lo que parezcan ser una variable de shell.
- Las tablas de cron reales normalmente definen variables de entorno antes de enumerar los comandos. Ignora esto.
- No preguntes al profesor qué errores debe detectar tu programa o qué errores hay en los ejemplos, eso forma parte de la materia del examen.
- Tu programa debería indicar todos los errores y avisos que encuentre. Sugerencia: escribe funciones para detectar posibles problemas sobre líneas de texto, y aplica todas las funciones a cada línea.

  Un enfoque diferente sería que en cuanto el programa encuentre un error o un aviso, lo muestre y deje de buscar. Esto es claramente peor. Procura evitarlo.

Tablas de ejemplo

```
# tabla01.txt
#   m    h   dayofmonth  month  dow   command
    *    *        *        *     *   /usr/bin/touch /tmp/probando.txt
    */5   *        *        *     *   $HOME/bin/holamundo.sh

# tabla02.txt
#   m    h   dayofmonth  month  dow   command
    *    *        *   *    *     *   /usr/bin/touch /tmp/probando.txt

# tabla03.txt
#   m    h   dayofmonth  month  dow   command
    *   *        *        *     *   /NADA/NADA/touch /tmp/probando.txt

# tabla04.txt
#   m    h   dayofmonth  month  dow   command
    *   *        *        14     9   /usr/bin/touch /tmp/probando.txt

# tabla05.txt
#   m    h   dayofmonth  month  dow   command
    *   *        *        1     3    touch /tmp/probando.txt

# tabla06.txt
#   m    h   dayofmonth  month  dow   command
    */5           *        4     *   $CASA/bin/holamundo.sh

// tabla07.txt
//  m    h   dayofmonth  month  dow   command
    *   8         *       *     1-5   /usr/bin/touch /tmp/probando.txt


# tabla08.txt
#   m    h   dayofmonth  month  dow   command
    *   22-7        *       *     1-5   /usr/bin/touch /tmp/probando.txt
```

Laboratorio de administración y gestión de redes y sistemas
Examen de teoría
4 de febrero de 2021.

Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Ejercicio 1. (3 puntos)

Describe brevemente cómo se puede aplicar el multicast en el clonado de máquinas, cuáles son sus ventajas y cuáles sus inconvenientes.

Ejercicio 2. (3 puntos)

¿En qué se parece UTF-8 a Unicode? ¿En qué se diferencian?

Respuesta

Son dos términos muy relacionados, con frecuecia se consideran sinónimos pero no lo son. Unicode es un estandar que aparece en la década de 1990 para representar las letras y algunos otros símbolos como jeroglíficos o emoticonos de todos los lenguajes del mundo, naturales o artificiales. Podemos decir que Unicode es una gran tabla con algo menos de 150 000 entradas, a cada caracter le asigna un número.

Hay diversas formas de codificar Unicode, esto es, de representar ese número en un ordenador. UTF-8 es una de ellas, la más usada actualmente. Es retrocompatible con ASCII, el número de bytes que ocupa cada caracter es variable (menos bytes los más frecuentes, más bytes los más raros), y se auto-sincroniza (tiene un mecanismo para evitar que en un flujo de bytes haya ambigüedades, para saber si cierto byte pertenece a un nuevo símbolo o al símbolo anterior)

Ejercicio 3. (4 puntos)

Como seguramente sabes, `chmod u+s` y `chmod u-s` permiten quitar y poner el bit SUID a un fichero. ¿Qué significa esto? ¿Para qué se usa? ¿Por qué puede suponer un problema de seguridad?

Laboratorio de administración y gestión de redes y sistemas
Examen práctico
30 de junio de 2021.

Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

- Entra en tu cuenta del laboratorio y ejecuta

  `~mortuno/prepara`

- Esto creará los ficheros

  `~/lagrs.julio.21/final.txt`

  `~/lagrs.julio.21/conexiones.py`

  Contesta las preguntas en estos dos ficheros, en ningún otro sitio.

Ejercicio 1 (6 puntos)

En este ejercicio indicarás todo lo necesario para preparar un contenedor docker según la siguiente especificación:

1. Deseamos que al ponerse en marcha el contenedor, el usuario *root* tenga una sesión de shell abierta.

2. El usuario podrá ejecutar la orden `maquina`. Será un script de shell que simplemente escriba en la salida estándar el mensaje *Estás en TU-HOST*, donde *TU-HOST* será el nombre del *host*, obtenido de la variable de entorno correspondiente [1].

3. El *path* completo del script *maquina* será `/opt/examen/bin/maquina`.

4. Como seguramente sabes, para que el usuario pueda ejecutar la orden tecleando `maquina` (y no un trayecto completo como `/opt/examen/bin/maquina`), tendrás que añadir cierta línea al fichero `.bashrc` del usuario *root*.

   Haz esto de forma similar a lo que hiciste en la práctica 1.20 para añadir una línea a `/etc/host`. Esto es:

   a) Prepara, en el directorio contexto, un fichero `delta_bashrc` que contenga la línea adecuada.

   b) Haz que este fichero aparezca en el directorio `/tmp/` de la imagen.

   c) Haz que cuando se prepare la imagen del contenedor, se añada el contenido de `delta_bashrc` al fichero `.bashrc` del usuario *root*.

5. El contenedor estará construido a partir de ubuntu 20.04, con los paquetes actualizados a la última versión disponible. No tendrá ningún otro paquete ni ninguna configuración adicional.

[1] En el caso particular de este examen, el nombre del host lo conoces a priori, será jperex2021-1. Pero no lo escribas como una cadena constante en el script. Obtenlo de la variable de entorno

6. La imagen del contenedor se llamará `ex2021`. Los contenedores basados en esta imagen (en este caso solo uno), tendrán como prefijo parte de tu nombre de usuario, y como sufijo, un número. Si tu nombre de usuario fuera *jperez*, el prefijo sería *jper*. El sufijo, *-1*. Por tanto el contenedor se llamaría `jperex2021-1`. El nombre de *host* será este mismo. Sustituye *jper* por los primeros 4 caracteres de tu nombre de usuario.

7. Siguiendo un convenio como el que hemos venido usando durante toda la asignatura, los nombres de los ficheros serían:

```
~/lagrs.jul.21/ex2021/construye.sh
~/lagrs.jul.21/ex2021/lanza_jperex2021-1.sh
~/lagrs.jul.21/ex2021/context/Dockerfile
~/lagrs.jul.21/ex2021/context/entrypoint.sh
~/lagrs.jul.21/ex2021/context/maquina
~/lagrs.jul.21/ex2021/context/delta_bashrc
```

   (Reemplazando *jper* por las primeras 4 letras de tu login). Pero no escribas en esos ficheros, **escribe solamente en final.txt** indicando qué habría en cada fichero. P.e. *Contenido del fichero X: blabla. Contenido del fichero Y: blabla, etc.*

Ejercicio 2 (4 puntos)

Escribe un programa en python llamado `~/lagrs.jul.21/conexiones.py` según la siguiente especificación:

1. El programa usará la librería *optparse* para aceptar un número de puerto, con la opción `-p`. Ejemplo: `conexiones.py -p 631`

2. Partiendo de la información proporcionada por la orden de shell `netstat -tupan`, el programa mostrará en la salida estándar todas las conexiones tcp v4 y todas las *pseudo-conexiones* udp v4 que usen dicho puerto en la dirección local. El programa ignorará las conexiones tcp v6 y las udp v6.

3. También indicará el número de conexiones totales que muestra.

En otras palabras, el programa invocará la orden `netstat -tupan` (con estas opciones y no otras), ignorarás las líneas correspondientes a *tcp6* y *udp6*, mostrará todas aquellas líneas cuyo puerto de la columna *Dirección local* sea el indicado, e indicará cuántas líneas son.

Laboratorio de administración y gestión de redes y sistemas
Examen de teoría
30 de junio de 2021.

Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

- Entra en tu cuenta del laboratorio y ejecuta

  `~mortuno/prepara`

- Esto creará el fichero `~/lagrs.julio.21/teoria.TULOGIN.txt` (donde TULOGIN es tu nombre de usuario). Contesta las cuatro preguntas en este fichero.

Ejercicio 1. (2.5 puntos)

En administración de sistemas Unix, cuando usamos la palabra *root* podemos referirnos a tres cosas distintas. Por escrito queda un poco más claro, de palabra puede haber confusiones. ¿Qué tres significados tiene?

Ejercicio 2. (2.5 puntos)

Explica brevemente en qué consiste el método *secret sharing* para la gestión de contraseñas, cuáles son sus ventajas y qué herramienta podemos usar en Linux con este propósito. No es necesario que indiques los detalles del uso de la herramienta.

Ejercicio 3. (2.5 puntos)

Explica brevemente el concepto *desarrollo de software ágil*.

Ejercicio 3. (2.5 puntos)

Cuando usamos *cron* hay que tener cuidado con las variables de entorno, es frecuente que el principiante cometa errores relacionados con esto. Explícalo brevemente.

Laboratorio de administración y gestión de redes y sistemas
Examen práctico
20 de enero de 2022.

Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Ejercicio 1 (6 puntos)

En este ejercicio prepararás un contenedor Docker según la siguiente especificación:

1. Deseamos que al ponerse en marcha el contenedor, el usuario *root* tenga una sesión de shell abierta.

2. En el contenedor, habrá un script de shell llamado `/opt/examen/bin/maquina`

3. Este script `maquina` escribirá en la salida estándar el mensaje *Estás en TU-HOST*, donde *TU-HOST* será el nombre del *host*, obtenido de la variable de entorno correspondiente.

   Naturalmente, *TU-HOST* será siempre el nombre de la máquina donde se ejecuta el script `maquina`. Cuando se ejecute en tu puesto del laboratorio, devolverá el nombre del puesto del laboratorio. Cuando se ejecute en el servidor de contenedores, devolverá el nombre del servidor de contenedores. Cuando se ejecute en un contenedor, devolverá el nombre de ese contenedor.

4. Dentro del contenedor, con la sesión de shell de root abierta, el usuario podrá ejecutar la orden `maquina`, *a secas*, sin necesidad de escribir el trayecto completo

   (`/opt/examen/bin/maquina`).

   Como seguramente sabes, para poder conseguir esto, tendrás que añadir cierta línea al fichero `.bashrc` del usuario *root*.

   Hazlo de forma similar a lo que hiciste en la práctica 1.23 para añadir una línea a `/etc/host`. Esto es:

   a) Prepara, en el directorio contexto, un fichero `delta_bashrc` que contenga la línea adecuada.

   b) Haz que este fichero aparezca en el directorio `/tmp/` de la imagen.

   c) Haz que cuando se prepare la imagen del contenedor, se añada el contenido de `delta_bashrc` al fichero `.bashrc` del usuario *root*.

5. El contenedor estará construido a partir de ubuntu 20.04, con los paquetes actualizados a la última versión disponible. No tendrá ningún otro paquete ni ninguna configuración adicional.

6. La imagen del contenedor se llamará `ex2022`. Los contenedores basados en esta imagen (en este caso solo uno), tendrán como prefijo parte de tu nombre de usuario, y como sufijo, un número. Si tu nombre de usuario fuera *jperez*, el prefijo sería *jper*. El sufijo, *-1*. Por tanto el contenedor se llamaría `jperex2022-1`. El nombre de *host* será este mismo. Sustituye *jper* por los primeros 4 caracteres de tu nombre de usuario.

7. Siguiendo un convenio como el que hemos venido usando durante toda la asignatura, los nombres de los ficheros serán:

```
~/lagrs.ene.22/ex2022/construye.sh
~/lagrs.ene.22/ex2022/lanza_jperex2022-1.sh
~/lagrs.ene.22/ex2022/context/Dockerfile
~/lagrs.ene.22/ex2022/context/entrypoint.sh
~/lagrs.ene.22/ex2022/context/maquina
~/lagrs.ene.22/ex2022/context/delta_bashrc
```

(Reemplazando *jper* por las primeras 4 letras de tu login).

Cuando acabes, enseña el resultado al profesor.

Ejercicio 2 (4 puntos)

Escribe un programa en python llamado `~/lagrs.ene.22/conexiones.py` según la siguiente especificación:

1. El programa usará la librería *optparse* para aceptar un número de puerto, con la opción `-p`. Ejemplo: `conexiones.py -p 631`
2. Partiendo de la información proporcionada por la orden de shell `netstat -tupan`, el programa mostrará en la salida estándar todas las conexiones tcp v4 y todas las *pseudo-conexiones* udp v4 que usen dicho puerto en la dirección local. El programa ignorará las conexiones tcp v6 y las udp v6.
3. También indicará el número de conexiones totales que muestra.

En otras palabras, el programa invocará la orden `netstat -tupan` (con estas opciones y no otras), ignorarás las líneas correspondientes a *tcp6* y *udp6*, mostrará todas aquellas líneas cuyo puerto de la columna *Dirección local* sea el indicado, e indicará cuántas líneas son.

Cuando acabes, enseña el resultado al profesor.

Laboratorio de administración y gestión de redes y sistemas
Examen de teoría
20 de enero de 2022.

Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

- Entra en tu cuenta del laboratorio y ejecuta

  `~mortuno/prepara`

  Comprueba que esto deja en tu cuenta el fichero `~/lagrs.enero.22/teoria.TULOGIN.txt` (donde TULOGIN es tu nombre de usuario).

- Contesta ahí las cuatro preguntas siguientes:

Ejercicio 1. (2.5 puntos)

En administración de sistemas Unix, cuando usamos la palabra *root* podemos referirnos a tres cosas distintas. Por escrito queda un poco más claro, de palabra puede haber confusiones. ¿Qué tres significados tiene?

Ejercicio 2. (2.5 puntos)

Explica brevemente en qué consiste el método *secret sharing* para la gestión de contraseñas, cuáles son sus ventajas y qué herramienta podemos usar en Linux con este propósito. No es necesario que indiques los detalles del uso de la herramienta.

Ejercicio 3. (2.5 puntos)

Explica brevemente el concepto *desarrollo de software ágil*.

Ejercicio 4. (2.5 puntos)

Cuando usamos *cron* hay que tener cuidado con las variables de entorno, es frecuente que el principiante cometa errores relacionados con esto. Explícalo brevemente.

Laboratorio de administración y gestión de redes y sistemas
Examen práctico. 18 de junio de 2022.
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Preparativos:

- Si no has hecho el examen de teoría, crea un directorio `~/lagrs.junio.22/`, donde escribirás todo lo indicado a continuación.

Ejercicio 1 (5 puntos)

En este ejercicio escribirás un script en python 3 que hará lo mismo que tu práctica 2.8, `~/lagrs/practica02/recientes.py`, con la siguiente diferencia:

1. El programa se llamará `~/lagrs.junio.22/bot.py`
2. El programa leerá de un fichero de texto con nombre `id_usuario.txt` el identificador de usuario de telegram que recibirá las alarmas. No especifiques ningún *path*, de forma que se busque en el directorio actual.
3. El programa leerá el token del bot de telegram en el fichero `token.txt`. De nuevo, no especifiques ningún trayecto, que el script lea el fichero desde el directorio actual.
4. El intervalo de tiempo no se expresará en días, sino en horas.
5. En tu práctica probablemente habrás usado la función `os.path.getmtime`. En este caso, modifícala para que se base en la orden de shell `ls -l --full-time`. Y viceversa: si en tu práctica usaste `ls`, ahora usa `os.path.getmtime`.
6. Los parámetros de entrada (nombre de directorio e intervalo de tiempo) no se los preguntará el bot al usuario, sino que los marcará el administrador como argumentos de la línea de comandos, al lanzar el script *bot.py*. Usa la librería *optparse*, como en tu práctica 2.4.

Ejercicio 2 (5 puntos)

En este ejercicio prepararás un contenedor docker según la siguiente especificación

1. En la imagen del contenedor habrá un fichero llamado `practicas.tgz`. La ubicación de este fichero dentro de la imagen será la que tu decidas. Procura que sea *adecuada*. Este fichero contendrá tus ficheros de de memoria de prácticas:

   ```
   ~/lagrs/practica01.md
   ~/lagrs/practica03.txt
   ~/lagrs/practica04.txt
   ```

2. Cada vez que se inicie el contenedor, estos ficheros deben estar disponibles, descomprimidos, dentro del directorio

   ```
   /root/practicas
   ```

3. En el contenedor deberá estar disponible el editor *vim*

4. Cuando tengas todo listo, enséñaselo al profesor. El profesor te pedirá que, usando vim, edites cada uno de los ficheros de de memoria de prácticas escribiendo en la primera línea tu nombre, apellidos y la fecha de hoy.

El contenedor estará construido a partir de ubuntu 20.04, con la configuración de idioma por omisión (inglés). Si añades paquetes, ficheros o configuración adicional, tendrás una penalización en la nota. Los ficheros necesarios para esto seguirán el mismo convenio que hemos venido usando durante toda la asignatura.

1. El contenedor estará basado en una imagen llamada `exa`. Si tu nombre de usuario fuera *jperez*, el contenedor se llamaría `jperexa01`. Sustituye *jper* por los primeros 4 caracteres de tu nombre de usuario.

2. Los nombres de los ficheros que debes preparar para esto serán, entre otros:

   ```
   ~/lagrs.junio.22/exa/construye.sh
   ~/lagrs.junio.22/exa/lanza_jperexa01.sh
   ~/lagrs.junio.22/exa/context/Dockerfile
   ~/lagrs.junio.22/exa/context/entrypoint.sh
   ```

   (Reemplazando *jper* por las primeras 4 letras de tu login) Esta lista no está completa, para cumplir con la especificación puedes necesitar algún fichero adicional

Laboratorio de Administración y Gestión de Redes y Sistemas
Examen de teoría. 18 de junio de 2022
Grado en Ingeniería Telemática, Universidad Rey Juan Carlos

Preparativos

- Ejecuta el script `~mortuno/prepara`
- Esto creará en tu ordenador el fichero `~/lagrs.junio.22/teoria.TULOGIN.txt`, donde TULOGIN es tu nombre de usuario en el laboratorio. Escribe tus respuestas en este fichero.

Ejercicio 1 (2.5 puntos)

¿Cuál es la diferencia entre las órdenes Unix *sudo* y *su*?

Ejercicio 2 (2.5 puntos)

Explica con detalle esta sesión de Unix: qué es cada linea, cada órden, qué hace, qué sucede y por qué.

```
01 koji@mazinger:~$ echo hola > fichero.txt
02 koji@mazinger:~$ file fichero.txt
03 fichero.txt: ASCII text
04 koji@mazinger:~$ echo mañana >> fichero.txt
05 koji@mazinger:~$ file fichero.txt
06 fichero.txt: UTF-8 Unicode text
```

(Evidentemente los números iniciales no forman parte de las órdenes, solo aparecen para facilitar la referencia)

Ejercicio 3 (2.5 puntos)

Describe brevemente qué es el *despliegue frecuente* y cuáles son sus principales ventajas.

Ejercicio 4 (2.5 puntos)

Describe brevemente qué es y para qué sirve un *túnel ssh inverso*. No es necesario que indiques detalles sobre parámetros y sintaxis, basta una descripción conceptual.

http://hdl.handle.net/10115/20116