Universidad Nacional de La Matanza
Florencio Varela 1903 - San Justo - Buenos Aires - Argentina

Departamento de Ingeniería e Investigaciones Tecnológicas

Cátedra de Virtualización de Hardware (3654)

Jefe de Cátedra:	**Alexis Villamayor**
Docentes:	**Alexis Villamayor, Fernando Boettner**
Jefe de trabajos prácticos:	**Ramiro de Lizarralde**
Ayudantes:	**Alejandro Rodriguez, Fernando Piubel**

Año: **2025 - Primer cuatrimestre**

Actividad Práctica de Laboratorio Nº 1
Bash y Powershell

Condiciones de entrega

- Se debe entregar por plataforma MIEL un archivo con formato ZIP o TAR (no se aceptan RAR u otros formatos de compresión/empaquetamiento de archivos), conteniendo la carátula que se publica en MIEL junto con los archivos de la resolución del trabajo.
- Se debe entregar el código fuente de cada uno de los ejercicios resueltos tanto en Bash como en Powershell. Si un ejercicio se resuelve en un único lenguaje se lo considerará incompleto y, por lo tanto, desaprobado.
- Se deben entregar lotes de prueba válidos para los ejercicios que reciban archivos o directorios como parámetro.
- Los archivos de código deben tener un encabezado en el que se listen los integrantes del grupo.
- Los archivos con el código de cada ejercicio y sus lotes de prueba se deben ubicar en un directorio con la siguiente estructura:
    - APL1/
        - bash/
            - ejercicio1
            - ejercicio2
            - ejercicio3
            - ejercicio4
            - ejercicio5
        - powershell/
            - ejercicio1
            - ejercicio2
            - ejercicio3
            - ejercicio4
            - ejercicio5

Criterios de corrección y evaluación generales para todos los ejercicios

- Los scripts de bash muestran una ayuda con los parámetros “-h” y “--help”. Deben permitir el ingreso de parámetros en cualquier orden, y no por un orden fijo.
- Los scripts de Powershell deben mostrar una ayuda con el comando Get-Help. Ej: “Get-Help ./ejercicio1.ps1”. Deben realizar la validación de parámetros en la sección params utilizando la funcionalidad nativa de Powershell.
- Cuando haya parámetros que reciban rutas de directorios o archivos se deben aceptar tanto rutas relativas como absolutas o que contengan espacios.
- No se debe permitir la ejecución del script si al menos un parámetro obligatorio no está presente.
- Si algún comando utilizado en el script da error, este se debe manejar correctamente: detener la ejecución del script (o salvar el error en caso de ser posible) y mostrar un mensaje informando el problema de una manera amigable con el usuario, pensando que el usuario **no** tiene conocimientos informáticos.
- Si se generan archivos temporales de trabajo se deben crear en el directorio temporal /tmp; y se deben eliminar al finalizar el script, tanto en forma exitosa como por error, para no dejar archivos basura. (Ver trap en bash / try-catch-finally en powershell)
- Deseable:
    - Utilización de funciones en el código para resolver los ejercicios.

Ejercicio 1

Objetivos de aprendizaje: manejo de archivos CSV, manejo de parámetros y salida por pantalla

Se quiere automatizar el cálculo de temperatura registrado en las diferentes estaciones climáticas ubicadas en la provincia de Buenos Aires. Para poder realizar esto, existe un proceso servidor que registra las mediciones que envía cada estación y las registra en un archivo para mantener el registro diario. Cada una de estas estaciones envía periódicamente a cada hora la medición de la temperatura y debido a que puede haber problemas de conexión, puede que el mensaje nunca llegue al servidor.
Tenga en cuenta que el servidor genera los archivos por día, indicando en el nombre del archivo la fecha de procesamiento y puede suceder que haya recibido un mensaje de otro día por demoras en la transmisión o error en el dispositivo.

El registro que genera el servidor tiene el siguiente formato:

- Id del dispositivo (numérico)
- Fecha de registro (texto: dd/mm/yyyy)
- Horade registro (texto: HH/MM/SS)
- Ubicación (texto Norte, Sur, Este, Oeste)
- Temperatura (numérico con decimales, ej: 21.2)

Por ejemplo:

2025-04-02.csv

1,2025/04/02,01:01:12,Sur,5.4
2,2025/04/02,01:01:12,Sur,5.9
3,2025/04/01,23:59:59,Norte,15.4

En un determinado momento se necesita relizar el procesamiento de toda la información recolectada y al finalizar el procesamiento, se indicará por pantalla en formato JSON los promedios, máximos y mínimos de temperatura agrupados por día y ubición.

```
{
    “fechas": [
        "2025-04-01": {
            "Sur": {
        “Min”: 2.3,
        “Max”: 32.3,
        “Promedio”: 22.3,
            },
            "Este": {
        “Min”: 3.3,
        “Max”: 31.3,
        “Promedio”: 21.3,
            },
            "Norte": {
        “Min”: 5.3,
        “Max”: 35.3,
        “Promedio”: 27.3,
            },
            "Oeste": {
```

```
            "Min": 5.3,
            "Max": 29.3,
            "Promedio": 20.3,
                }
            }
        "2025-04-02": {
        ...
        },
        ....
        ]
}
```

Consideraciones:

1. El formato del JSON tiene que ser válido.
2. Los resultados se pueden mostrar tanto por pantalla como en un archivo, pero solo se puede generar 1 de las 2 formas en una ejecución.
3. Los archivos no tienen encabezado.

Parámetros:

Parámetro bash	Parámetro PowerShell	Descripción
-d / --directorio	-directorio	Ruta del directorio que contiene los archivos CSV a procesar.
-a / --archivo	-archivo	Ruta del archivo JSON de salida (no es un directorio, es la ruta completa incluyendo el nombre del archivo). Este parámetro no se puede usar a la vez que –p o -pantalla.
-p / --pantalla	-pantalla	Muestra la salida por pantalla, no genera el archivo JSON. Este parámetro no se puede usar a la vez que –a o -archivo.

Ejercicio 2

Objetivos de aprendizaje: arrays y matrices.

Desarrollar un script que permita realizar producto escalar y trasposición de matrices. La entrada de la matriz al script se realizará mediante un archivo de texto plano. La salida se guardará en otro archivo que se llamará “salida.nombreArchivoEntrada” y se generará en el mismo directorio donde se encuentre el archivo de la matriz que se está procesando (puede ser cualquier directorio, considerar que no necesariamente es donde se ejecuta el script).

El formato de los archivos de matrices debe ser el siguiente:

```
0|1|2
1|1|1
-3|-3|-1
```

Consideraciones:

1. Tener en cuenta que los archivos pueden contener matrices inválidas, esto puede ser porque la cantidad de columnas por fila no coincidan o que contengan algún valor que no sea numérico o incluso ser un archivo vacío.
2. El carácter separador puede ser cualquier carácter salvo números y el símbolo menos (“-”) para no confundir con los números negativos (esto debe ser parte de las validaciones del script).
3. Los valores de la matriz serán numéricos. Esto incluye valores decimales y negativos.

Parámetros:

Parámetro bash	**Parámetro PowerShell**	**Descripción**
-m / --matriz	-matriz	Ruta del archivo de la matriz.
-p / --producto	-producto	Valor entero para utilizarse en el producto escalar. No se puede usar junto con -t o -trasponer
-t / --trasponer	-trasponer	Indica que se debe realizar la operación de trasposición sobre la matriz. (no recibe valor adicional, solo el parámetro) No se puede usar junto a –p o –producto.
-s / --separador	-separador	Carácter para utilizarse como separador de columnas.

Ejercicio 3

Objetivos de aprendizaje: arrays asociativos, búsqueda de archivos, manejo de archivos, AWK

Desarrollar un script que identifique la cantidad de ocurrencias de determinadas palabras en determinados archivos dentro de un directorio (incluyendo los subdirectorios). Para esto, se reciban por parámetro tanto la lista de palabras a buscar, como las extensiones de archivos dónde buscar dichas palabras.

La salida del script debe ser un listado ordenado de las palabras con el contador de veces que fue detectada.

Ejemplo de salida:

if: 10
for: 9
else: 5

Consideraciones:

1. En bash el procesamiento de la información se realizará mediante AWK y queda a elección de los grupos cómo enviarle el contenido de los archivos identificados a AWK.
2. En Powershell, la lista de palabras y extensiones debe ser un objeto de tipo lista.
3. Tener en cuenta que las búsquedas deber ser case sensitive.

Parámetros:

Parámetro bash	Parámetro PowerShell	Descripción
-d / --directorio	-directorio	Ruta del directorio a analizar.
-p / --palabras	-palabras	Lista de palabras a contabilizar.
-a / --archivos	-archivos	Lista de extensiones de archivos a buscar.

Ejercicio 4

Objetivos de aprendizaje: procesos demonios, monitoreo de directorios, herramientas de compresión y archivado

Un amigo suyo, muy ordenado y un poco obsesivo, tuvo la idea de organizar su carpeta de descargas por extensión. De esta manera, cada vez que descarga un nuevo archivo (es decir, un archivo es creado o copiado a la carpeta de descargas), lo busca y manualmente lo mueve a un directorio que tiene por nombre la extensión de dicho archivo. Por ejemplo, un archivo .zip irá a la carpeta “ZIP” y uno .pdf irá a “PDF” y como usted es una buena persona, decide inventar una solución para automatizar la tarea que realiza su amigo.

Se pide realizar un script demonio que detecte cada vez que un archivo nuevo aparezca en un directorio “descargas”. Una vez detectado, se debe mover a un subdirectorio “extensión” cuyo nombre será la extensión del archivo y que estará localizado en un directorio “destino”. Por un tema de seguridad durante las pruebas, cada X cantidad de ordenamiento de archivos, se genera un backup del directorio “descargas” en caso de tener un problema y eliminar los archivos. Este. Backup debe tener el nombre del directorio que se está respaldando, junto a la fecha y hora (yyyyMMdd-HHmmss),
Por ejemplo: ***descargas_20250401_212121.zip.***

Tener en cuenta que, al ser un demonio, el script debe liberar la terminal una vez ejecutado, dejando al usuario la posibilidad de ejecutar nuevos comandos.

Consideraciones:

1. El script debe quedar ejecutando por sí solo en segundo plano, el usuario no debe necesitar ejecutar ningún comando adicional a la llamada del propio script para que quede ejecutando como demonio en segundo plano. La solución debe utilizar un único archivo script (por cada tecnología), no se aceptan soluciones con dos o más scripts que trabajen en conjunto.
2. El script debe poder ejecutarse nuevamente para finalizar el demonio ya iniciado. Debe validar que esté en ejecución sobre el directorio correspondiente.
3. No se debe poder ejecutar más de 1 proceso demonio para un determinado directorio al mismo tiempo.
4. El monitoreo del directorio se debe hacer utilizando inotify-tools en bash y FileSystemWatcher en Powershell.
5. El proceso demonio debe ejecutar su funcionalidad para los archivos existentes en el directorio y luego quedar a la espera de los nuevos para volver a ejecutar.

Ejemplo de uso:
$ ./demonio.sh -d ../descargas --backup ../backup -c 3

$ ./demonio.sh -d ../documentos --backup ../backup --cantidad 3

> ./demonio.ps1 -directorio ../descargas -backup ../backup -cantidad 3
> ./demonio.ps1 -directorio ../documentos -backup ../backup -cantidad 3

$ ./demonio.sh -d ../descargas --kill
> ./demonio.ps1 -directorio ../descargas -kill

Parámetro bash	**Parámetro PowerShell**	**Descripción**
-d / --directorio	-directorio	Ruta del directorio a monitorear
-s / --salida	-salida	Ruta del directorio en donde se van a crear los backups.
-k / --kill	-kill	Flag (switch) que se utiliza para indicar que el script debe detener el demonio previamente iniciado. Este parámetro solo se puede usar junto con –d o -directorio.
-c / --cantidad	-cantidad	Cantidad de archivos a ordenar antes de generar un backup.

Ejercicio 5
Objetivos de aprendizaje: conexión con APIs y web services, manejo de archivos y objetos JSON, cache de información

Se necesita implementar un script que facilite la consulta de información relacionada a los nutrientes de las frutas para poder diagramar una dieta. El script permitirá buscar información de las frutas id o nombre a través de la api Fruityvice y pueden enviarse más de un valor tanto para los ids como los nombres en la ejecución del script e incluso solicitar la búsqueda por ambos parámetros (tener en cuenta que los nombres están en inglés).
Una vez obtenida la información, se generará a modo de caché, para evitar volver a consultarlo a la api, y se mostrará por pantalla la información con el siguiente formato:

(uno por cada resultado encontrado)
id: 2,
name: Orange,
genus: Citrus,
calories: 43,
fat: 0.2,
sugar: 8.2,
carbohydrates: 8.3,
protein: 1

La información de cómo obtener los datos se puede consultar en el siguiente link:

https://www.fruityvice.com/

Por ejemplo:
https://www.fruityvice.com/api/fruit/2
https://www.fruityvice.com/api/fruit/banana

En caso de ingresar un id inválido, se deberá informar al cliente un mensaje acorde. Mismo en caso de que la api retorne un error.

Ejemplo de uso:
Bash:
$ ./api.sh --id "11,22" --name "banana,orange"

PS:
> ./api.ps1 -id 11,22 –name “banana, orange”

Consideraciones:

1. Los parámetros al utilizar Powershell deben ser de tipo array, es decir, no es correcto que sea un string y luego utilizar una función como “.split(‘,’)” para obtener los valores correspondientes.

Parámetros:

Parámetro bash	**Parámetro PowerShell**	**Descripción**
-i / --id	-id	Id/s de las frutas a buscar.
-n / --name	-name	Nombre/s de las frutas a buscar.