Universidad de San Carlos de Guatemala
Escuela de Ciencias Físicas y Matemáticas
Laboratorio de Simulación
Laboratorio 3: Docker, Bash, Git

Introducción

Docker es una tecnología de creación de contenedores que permite la creación y el uso de contenedores de Linux, puede usar los contenedores como máquinas virtuales extremadamente livianas y modulares. En este laboratorio se dan las bases para poder utilizar Docker con una imagen de Linux, la automatización de tareas creadas por scripts de Bash y el manejo de versiones por medio de la herramienta Git.

Objetivos

1. Conocer la funcionalidad de Docker y sus ventajas ante maquinas virtuales.
2. Crear un Dockerfile para la configuracion inicial de Docker.
3. Realizar scripts con condicionantes para el entorno Bash.
4. Crear un repositorio de Git para utilizarlo durante el curso.

Requerimientos

- Computador con sistema operativo Windows, Linux o MACOS y conexión a internet.
- **Visual Studio Code**
- Si esta utilizando windows necesita tener activado **WSL2**.
- **Docker**
- Cuenta en **GitHub**.

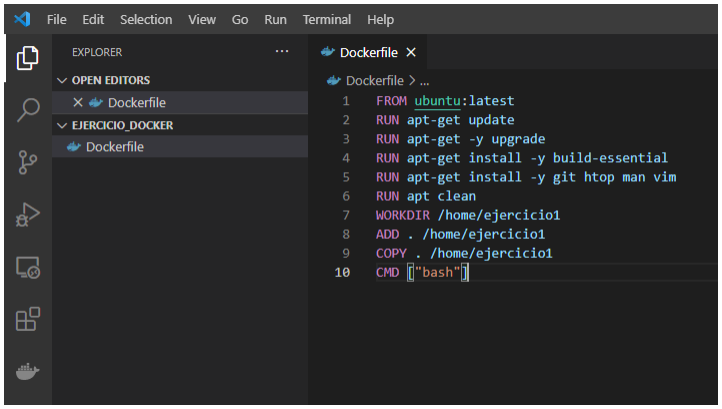
Docker

Al tener instalado Docker debe correr desde una terminal el ejemplo de **Hola mundo**. Debe documentar los comandos que utilizo para que el ejercicio se ejecute bien, recuerde este le devolverá esta respuesta.

```
¡Hola de DockerCon EU 2015 (Barcelona)!
This message shows that your installation appears to be working correctly.
```

Dockerfile

Crear el una carpeta llamada ejercicio_docker, abrir esta carpeta desde visual studio code y crear el Dockerfile con la siguiente configuración (recuerde debe de tener instalada la extension de docker en vs code).



Explique que es lo que hace cada linea de código. Guarde el archivo y construya su imagen de nombre ejdocker, para esto abrir la terminal desde vs code, recuerde de explicar que hace la siguiente linea de código

```
$ docker build -t ejdocker:latest .
```

Corra la imagen de docker y crear un archivo dentro de ubuntu, vea que sucede con los archivos que aparecen en la ventana de visual studio code, describa cada linea de código

```
$ docker run -it ejdocker bash
# touch archivo1
```

Salga del ubuntu y vuelva a correrlo pero con la siguiente linea de código, donde dice PATHLOCAL se refiere al pad donde usted creo la carpetaejercicio_docker, crear un archivo dentro de ubuntu, vea que sucede con los archivos que aparecen en la ventana de visual studio code, describa cada linea de código, por ultimo crear un archivo desde vs code y liste los archivos en windows para ver que sucede.

```
$ docker run -it -v PATHLOCAL:/home/ejercicio1 -w /home/ejercicio1 ejdocker bash
# touch archivo2
//crear un archivo desde el entorno visual de vscode de nombre archivonuevo
# ls
```

Describa las diferencias que existen entre las dos formas de ejecutar la imagen creada.

Bash Script:

Con los conocimientos previos de Bash, podemos realizar scripts que automaticen una tarea dentro de la consola de Linux. Creamos un script agregando la extensión **.sh** al nombre del archivo, NO OLVIDAR darle permisos de ejecución. *touch holamundo.sh && chmod +x holamundo.sh*. Podemos crear el script en vscode y correrlo en consola. La linea introductoria: `#!/bin/bash` siempre debe de ir al inicio del script.

```
File  Edit  View  Search  Terminal  Help
 1 #!/bin/bash
 2 #esto es un comentario
 3 nombre="mundo" #asi se define la variable igualandola a mundo
 4 echo "Hola $nombre" #$ antes de la varible sirve para usarla
 5 date #esto nos dara la fecha
```

```
File  Edit  View  Search  Terminal  Help
geotoj@iamgroot:~/Documents$ ./holamundo.sh
Hola mundo
lun feb 17 16:17:34 CST 2020
geotoj@iamgroot:~/Documents$
```

Entre los elementos que podemos utilizar témenos:

Variables y manipulación de texto

```
name="John"
echo ${name}
echo ${name/J/j}    #=> "john" (substitution)
echo ${name:0:2}    #=> "Jo" (slicing)
echo ${name::2}     #=> "Jo" (slicing)
echo ${name::-1}    #=> "Joh" (slicing)
echo ${name:(-1)}   #=> "n" (slicing from right
echo ${name:(-2):1} #=> "h" (slicing from right
echo ${food:-Cake}  #=> $food or "Cake"
```

```
length=2
echo ${name:0:length}  #=> "Jo"
```

See: Parameter expansion

```
STR="/path/to/foo.cpp"
echo ${STR%.cpp}    # /path/to/foo
echo ${STR%.cpp}.o  # /path/to/foo.o

echo ${STR##*.}     # cpp (extension)
echo ${STR##*/}     # foo.cpp (basepath)

echo ${STR#*/}      # path/to/foo.cpp
echo ${STR##*/}     # foo.cpp

echo ${STR/foo/bar} # /path/to/bar.cpp
```

```
STR="Hello world"
echo ${STR:6:5}   # "world"
echo ${STR:-5:5}  # "world"
```

```
SRC="/path/to/foo.cpp"
BASE=${SRC##*/}   #=> "foo.cpp" (basepath)
DIR=${SRC%$BASE}  #=> "/path/to/" (dirpath)
```

Condicionantes IF

Conditions

Note that `[[` is actually a command/program that returns either `0` (true) or `1` (false). Any program that obeys the same logic (like all base utils, such as `grep(1)` or `ping(1)`) can be used as condition, see examples.

`[[ -z STRING ]]`	Empty string
`[[ -n STRING ]]`	Not empty string
`[[ STRING == STRING ]]`	Equal
`[[ STRING != STRING ]]`	Not Equal
`[[ NUM -eq NUM ]]`	Equal
`[[ NUM -ne NUM ]]`	Not equal
`[[ NUM -lt NUM ]]`	Less than
`[[ NUM -le NUM ]]`	Less than or equal
`[[ NUM -gt NUM ]]`	Greater than
`[[ NUM -ge NUM ]]`	Greater than or equal
`[[ STRING =~ STRING ]]`	Regexp
`(( NUM < NUM ))`	Numeric conditions
`[[ -o noclobber ]]`	If OPTIONNAME is enabled
`[[ ! EXPR ]]`	Not
`[[ X ]] && [[ Y ]]`	And
`[[ X ]] || [[ Y ]]`	Or

File conditions

`[[ -e FILE ]]`	Exists
`[[ -r FILE ]]`	Readable
`[[ -h FILE ]]`	Symlink
`[[ -d FILE ]]`	Directory
`[[ -w FILE ]]`	Writable
`[[ -s FILE ]]`	Size is > 0 bytes
`[[ -f FILE ]]`	File
`[[ -x FILE ]]`	Executable
`[[ FILE1 -nt FILE2 ]]`	1 is more recent than 2
`[[ FILE1 -ot FILE2 ]]`	2 is more recent than 1
`[[ FILE1 -ef FILE2 ]]`	Same files

Ejemplos de if, note que en Bash el if finaliza con un **fi**

```
# String
if [[ -z "$string" ]]; then
  echo "String is empty"
elif [[ -n "$string" ]]; then
  echo "String is not empty"
fi

# Combinations
if [[ X ]] && [[ Y ]]; then
  ...
fi

# Equal
if [[ "$A" == "$B" ]]

# Regex
if [[ "A" =~ . ]]

if (( $a < $b )); then
   echo "$a is smaller than $b"
fi

if [[ -e "file.txt" ]]; then
  echo "file exists"
fi
```

Para esta sección debe de realizar dos ejercicios. Donde el puede hacerle una foto desde vscode o pegar el código en látex. **OJO** en látex existe una librería que nos permite agregar código, para evitar conflictos. Estos ejercicios debe trabajarlos dentro de la imagen de docker que recién creo y ejecuto, utilizando la segunda opción de ejecución de la imagen.

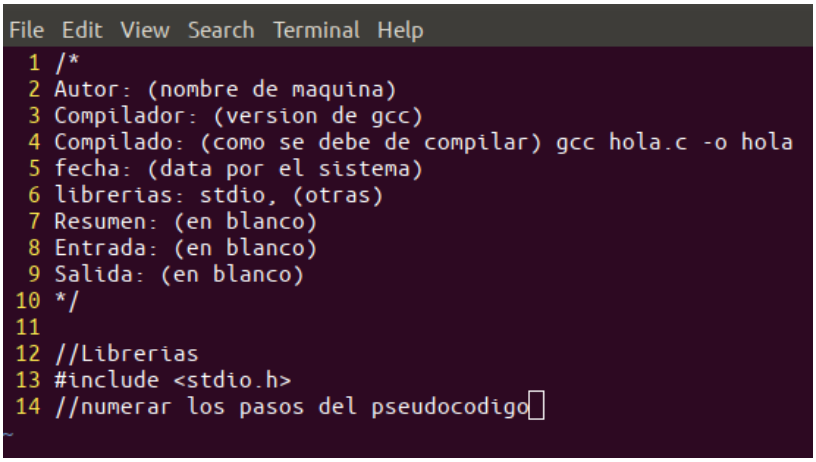
Ejercicio 1

Crear un script de nombre **clima** que nos muestre las 7 primeras lineas de la información obtenida, para esto:

- obtenga el clima con la instrucción **curl wttr.in/Guatemala**
- esta información la debe de almacenar en un archivo (no importa el nombre)
- leer del archivo las primeras 7 lineas
- escribir en consola lo leído
- borrar el archivo

Ejercicio 2

Crear un script que cree automáticamente un archivo para programar en c. Este debe de preguntar por el nombre del archivo, al escribirlo, el script debe de verificar si el nombre esta vació, si no existe tal nombre, el programa deberá terminarse y no generar archivos. En caso contrario, el programa debe de crear el archivo (nombre.c) y obtener de forma automática el nombre del usuario de Linux, la fecha, y la versión de gcc que esta instalada en el sistema. Debe de permitir ingresar el resumen, las entradas y salidas del programa. Para crear el siguiente header: Recuerde al terminar de escribir cada script debe darle permisos



de ejecución para que este pueda correr.

Git y GitHub:

Debe de configurar git desde consola:

```
$ git config --global user.name "su nombre"
$ git config --global user.email sucorreo
```

En su github debe crear un repositorio de nombre **LabSimu1S2021**, recuerde que este debe de contener el README y la licencia de MIT.

Clone este repositorio dentro del docker

```
$ git clone direccion_repositorio
```

Siga los siguientes pasos desde su terminal

- Ingrese a la carpeta del repo (LabSimu1S2021) y cree una nueva carpeta llamada Bash
- Mover los ejercicios 1 y 2 a la carpeta /LabSimu1S2021/Bash
- Crear un archivo elementos.txt
- Crear el archivo .gitignore y agregar dentro de el elementos.txt
- realizar un push a su directorio remoto.