



APRENDE **BASH** GUÍA PRÁCTICA

A C A D E M I A X

Contenido

1	Introducción	4
1.1	Bienvenida	4
1.1.1	Libro vivo	5
1.1.2	Alcance	5
1.2	Prerequisitos	5
1.3	¿Cómo evitar bloqueos?	6
2	Primeros pasos	7
2.1	Visión general	7
2.1.1	¿Qué es y por qué debes aprenderlo?	7
2.1.2	¿En dónde se utiliza?	7
2.1.3	¿Qué trabajos puedes conseguir?	8
2.1.4	¿Cuánto puedes ganar?	8
2.1.5	¿Cuales son las preguntas más comunes?	8
2.2	Historia, evolución, y versiones	9
2.3	Características y ventajas	10
2.4	Diferencias con otros lenguajes de programación	11
2.5	Configuración	11
2.5.1	IDE	11
2.5.2	Entorno	12
2.6	Hola Mundo	12
3	Gramática	13
3.1	Sintaxis	13
3.2	Comentarios	13
4	Tipos y variables	14
4.1	Variables	14
4.2	Listas	16

Contenido

4.3	Objetos	18
4.4	Constantes	20
5	Operadores	20
5.1	Operadores de aritméticos	20
5.2	Operadores comparativos	21
5.3	Operadores lógicos	22
6	Condicionales	23
6.1	Condición única	23
6.2	Múltiples condiciones	24
7	Bucles	26
7.1	Bucle for	26
7.2	Bucle while	27
8	Funciones	28
9	Módulos	31
10	Siguientes pasos	33
10.1	Herramientas	33
10.2	Recursos	33
10.3	¿Que viene después?	34
10.4	Preguntas de entrevista	35

1 Introducción

1.1 Bienvenida

Bienvenid@ a esta guía de Academia X en donde aprenderás Bash práctico.

Hola, mi nombre es Xavier Reyes Ochoa y soy el autor de esta guía. He trabajado como consultor en proyectos para Nintendo, Google, entre otros proyectos top-tier, trabajé como líder de un equipo de desarrollo por 3 años, y soy Ingeniero Ex-Amazon. En mis redes me conocen como Programador X y comparto videos semanalmente en YouTube desde diversas locaciones del mundo con el objetivo de guiar y motivar a mis estudiantes mientras trabajo haciendo lo que más me gusta: la programación.

En esta guía vas a aprender estos temas:

- Gramática
- Tipos y variables
- Operadores
- Condicionales
- Bucles
- Funciones
- Módulos

La motivación de esta guía es darte todo el conocimiento técnico que necesitas para elevar la calidad de tus proyectos y al mismo tiempo puedas perseguir metas más grandes, ya sea utilizar esta tecnología para tus pasatiempos creativos, aumentar tus oportunidades laborales, o si tienes el espíritu emprendedor, incluso crear tu propio negocio en línea. Confío en que esta guía te dará los recursos que necesitas para que tengas éxito en este campo.

Empecemos!

1 INTRODUCCIÓN

1.1.1 Libro vivo

Esta publicación fue planeada, editada, y revisada manualmente por Xavier Reyes Ochoa. La fundación del contenido fue generada parcialmente por inteligencia artificial usando ChatGPT (Mar 14 Version) de OpenAI. Puedes ver más detalles en <https://openai.com/>

Esto es a lo que llamo un trabajo **VIVO**, esto quiere decir que será actualizado en el tiempo a medida que existan cambios en la tecnología. La versión actual es 1.0.0 editada el 23 de marzo de 2023. Si tienes correcciones importantes, puedes escribirnos a nuestra sección de contacto en <https://www.academia-x.com>

1.1.2 Alcance

El objetivo de esta guía es llenar el vacío que existe sobre esta tecnología en Español siguiendo el siguiente enfoque:

- Se revizan los temas con un enfoque práctico incluyendo ejemplos y retos.
- Se evita incluir material de relleno ya que no es eficiente.
- Se evita entrar en detalle en temas simples o avanzados no-prácticos.

Si deseas profundizar en algún tema, te dejo varios recursos populares y avanzados en la lección de recursos como el estándar de esta tecnología (que puede ser difícil de leer si recién estás empezando).

1.2 Prerequisitos

Antes de aprender Bash, necesitas lo siguiente:

1. Un computador: cualquier computador moderno tiene las capacidades de correr este lenguaje. Te recomiendo un monitor de escritorio o laptop ya que dispositivos móviles o ipads no son cómodos para programar.

1 INTRODUCCIÓN

2. Sistema operativo: conocimiento básico de cómo utilizar el sistema operativo en el que se ejecutará Bash (por ejemplo, Windows, MacOS, Linux). Te recomiendo tener el sistema operativo actualizado.
3. Conocimiento básico de la línea de comandos: se utiliza para ejecutar programas en Bash.
4. Un editor de texto: lo necesitas para escribir código de Bash. Los editores de texto más populares son Visual Studio Code, Sublime Text, Atom y Notepad ++.
5. Un navegador web y conexión al internet: es útil para investigar más sobre esta tecnología cuando tengas dudas. Los navegadores más populares son Google Chrome, Mozilla Firefox, Safari y Microsoft Edge. Se recomienda tener el navegador actualizado.

Si ya tienes estos requisitos, estarás en una buena posición para comenzar a aprender Bash y profundizar en sus características y aplicaciones.

Si todavía no tienes conocimiento sobre algunos de estos temas, te recomiendo buscar tutoriales básicos en blogs a través de Google, ver videos en YouTube, o hacer preguntas a ChatGPT. Alternativamente, puedes empezar ya e investigar en línea a medida que encuentres bloqueos enteniendo los conceptos en esta guía.

1.3 ¿Cómo evitar bloqueos?

Para sacarle el mayor provecho a esta guía:

1. No solo leas esta guía. La práctica es esencial en este campo. Practica todos los días y no pases de lección hasta que un concepto esté 100% claro.
2. No tienes que memorizarlo todo, solo tienes que saber donde están los temas para buscarlos rápidamente cuando tengas dudas.
3. Cuando tengas preguntas usa [Google](#), [StackOverflow](#), y [ChatGPT](#)
4. Acepta que en esta carrera, mucho de tu tiempo lo vas utilizar investigando e innovando, no solo escribiendo código.

5. No tienes que aprender inglés ahora pero considera aprenderlo en un futuro porque los recursos más actualizados están en inglés y también te dará mejores oportunidades laborales.
6. Si pierdas la motivación, recuerda tus objetivos. Ninguna carrera es fácil pero ya tienes los recursos para llegar muy lejos. Te deseo lo mejor en este campo!

2 Primeros pasos

2.1 Visión general

2.1.1 ¿Qué es y por qué debes aprenderlo?

Bash es un intérprete de línea de comandos para sistemas operativos basados en Unix. Es una herramienta útil para automatizar tareas y realizar tareas administrativas en un sistema. Al aprender Bash, puedes ahorrar tiempo al automatizar tareas y realizar tareas administrativas de forma más eficiente. Además, puedes escribir scripts para realizar tareas repetitivas y complejas. Esto te permite ahorrar tiempo y esfuerzo al realizar tareas complejas.

2.1.2 ¿En dónde se utiliza?

Bash es un lenguaje de programación de scripting ampliamente utilizado en sistemas operativos basados en Unix, como Linux y macOS. Se utiliza para automatizar tareas comunes, como la ejecución de comandos, la creación y modificación de archivos y la administración de procesos. También se utiliza para escribir scripts para automatizar tareas complejas.

2.1.3 ¿Qué trabajos puedes conseguir?

- Desarrollador de software
- Administrador de sistemas
- Programador de scripts
- Analista de sistemas
- Desarrollador de aplicaciones web
- Desarrollador de aplicaciones de escritorio
- Desarrollador de aplicaciones móviles
- Desarrollador de aplicaciones de seguridad
- Desarrollador de aplicaciones de bases de datos
- Desarrollador de aplicaciones de red
- Desarrollador de aplicaciones de administración de sistemas

2.1.4 ¿Cuánto puedes ganar?

Dependiendo de la posición y el nivel de experiencia, los salarios para trabajos relacionados con Bash pueden variar. Por ejemplo, un programador de Bash con experiencia puede ganar entre \$60,000 y \$90,000 al año, mientras que un administrador de sistemas con experiencia puede ganar entre \$70,000 y \$110,000 al año.

Es importante tener en cuenta que estos son solo promedios y que el salario real que puedes ganar puede ser mayor o menor, dependiendo de los factores mencionados anteriormente. Además, siempre es una buena idea investigar y hacer preguntas sobre los salarios y las condiciones laborales antes de aceptar un trabajo.

2.1.5 ¿Cuales son las preguntas más comunes?

1. ¿Cómo ejecuto un script de Bash?
2. ¿Cómo puedo ver los comandos disponibles en Bash?
3. ¿Cómo puedo ver los argumentos de un comando en Bash?

4. ¿Cómo puedo ejecutar un comando en Bash con argumentos?
5. ¿Cómo puedo ejecutar un comando en Bash con variables?
6. ¿Cómo puedo ejecutar un comando en Bash con un bucle?
7. ¿Cómo puedo ejecutar un comando en Bash con una condición?
8. ¿Cómo puedo ejecutar un comando en Bash con una función?
9. ¿Cómo puedo ver los errores en Bash?
10. ¿Cómo puedo ver los archivos en Bash?

Después de completar este recurso, podrás responder a estas preguntas sin problemas.

Al finalizar este recurso, tendrás las habilidades necesarias para responder o encontrar las respuestas a estas preguntas.

2.2 Historia, evolución, y versiones

Bash (Bourne-Again Shell) es un intérprete de línea de comandos de Unix y una de las herramientas más populares para la administración de sistemas. Fue creado por Brian Fox para el Proyecto GNU en 1989 como un reemplazo para el shell original de Unix, el Bourne Shell.

Bash es una mezcla de los lenguajes de programación C y sh, y es compatible con la mayoría de los sistemas Unix, Linux y Mac OS X. Está diseñado para ser fácil de usar para los usuarios principiantes, pero también ofrece una gran cantidad de funcionalidades avanzadas para los usuarios más experimentados.

A lo largo de los años, Bash ha evolucionado para incluir una variedad de características nuevas y mejoras. La última versión estable de Bash es la 5.0, que fue lanzada en 2019. Esta versión incluye mejoras en la seguridad, el rendimiento y la compatibilidad con otros sistemas.

Además de la versión estable, hay también versiones de desarrollo de Bash. Estas versiones contienen características nuevas y mejoras que aún no han sido lanzadas

en la versión estable. Estas versiones de desarrollo se actualizan con frecuencia y se pueden descargar desde el sitio web del proyecto.

2.3 Características y ventajas

Bash es un intérprete de línea de comandos de Unix/Linux que se utiliza para ejecutar comandos y scripts. Está diseñado para ser fácil de usar y ofrece una gran cantidad de características y ventajas.

Características:

- Bash es un intérprete de línea de comandos completo que admite una amplia gama de comandos y herramientas.
- Ofrece una gran cantidad de funciones avanzadas, como la edición de líneas, la programación de scripts, la administración de archivos y la automatización de tareas.
- Bash es compatible con la mayoría de los sistemas Unix/Linux y se puede utilizar para ejecutar comandos en varios sistemas.
- Bash es un intérprete de línea de comandos fácil de usar y ofrece una gran cantidad de funciones útiles.

Ventajas:

- Bash es un intérprete de línea de comandos potente y fácil de usar.
- Ofrece una gran cantidad de funciones avanzadas para la edición de líneas, la programación de scripts, la administración de archivos y la automatización de tareas.
- Es compatible con la mayoría de los sistemas Unix/Linux y se puede utilizar para ejecutar comandos en varios sistemas.
- Es un intérprete de línea de comandos seguro y confiable.

2.4 Diferencias con otros lenguajes de programación

Bash es un lenguaje de programación de scripting que se utiliza para automatizar tareas. A diferencia de otros lenguajes de programación, Bash no es un lenguaje de programación orientado a objetos, sino un lenguaje de scripting. Esto significa que Bash no tiene clases, objetos, herencia, etc. Tampoco tiene una sintaxis estricta como otros lenguajes de programación. Esto significa que Bash es más fácil de aprender y usar.

Otra diferencia entre Bash y otros lenguajes de programación es que Bash está diseñado para interactuar con el sistema operativo. Esto significa que puede usar comandos del sistema operativo para realizar tareas. Esto hace que Bash sea útil para automatizar tareas como la administración de sistemas.

En general, Bash es un lenguaje de scripting útil para automatizar tareas y administrar sistemas. A diferencia de otros lenguajes de programación, Bash no es un lenguaje orientado a objetos y no tiene una sintaxis estricta. Esto lo hace más fácil de aprender y usar.

2.5 Configuración

2.5.1 IDE

Los archivos de Bash son archivos de texto. Puedes editarlos con **editores de texto** como Notepad en Windows o Notes en MacOS pero es recomendado utilizar un **IDE** (Integrated Development Environment) que es una aplicación de edición de código más avanzado que le da colores a tu código para que sea más fácil de leer y tengas funciones de autocompletado, entre otras. Algunos IDEs populares son [Brackets](#), [Atom](#), [Sublime Text](#), [Vim](#), y [Visual Studio Code](#).

El editor recomendado para practicar el código que vamos a ver es Visual Studio Code (o VSCode) que puedes bajar desde <https://code.visualstudio.com/>

2.5.2 Entorno

Para usar Bash en tu computador, primero debes tener una distribución de Linux. Algunas de las distribuciones más populares incluyen Ubuntu, Debian, Fedora y CentOS. Bash viene incluido en el sistema operativo macOS al abrir la terminal y escribir el comando **bash**.

Para verificar que tienes Bash puedes abrir terminal y escribir el comando **bash -version**. Esto mostrará la versión de Bash que estás usando.

Para correr archivos de Bash, primero debes asegurarte de que el archivo tenga la extensión **.sh**. Luego, puedes ejecutar el archivo desde la terminal usando el comando **./nombre_del_archivo.sh**. Si no ves el archivo, puedes convertirlo en ejecutable con el comando **chmod 777 ./nombre_del_archivo.sh**.

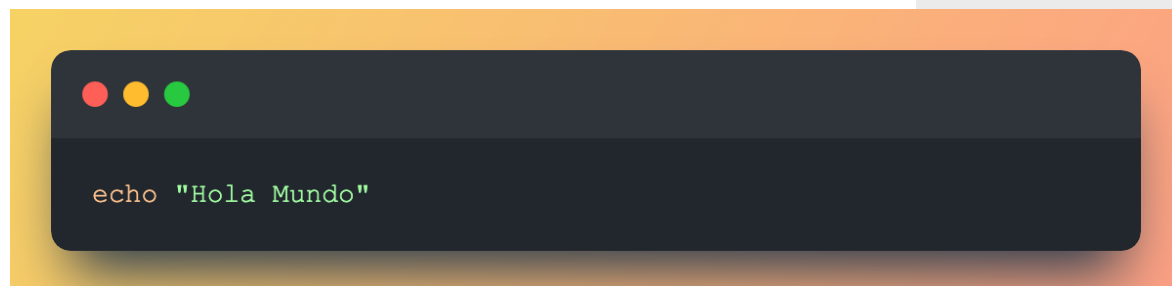
Para obtener más información sobre Bash, puedes consultar la documentación oficial aquí: <https://www.gnu.org/software/bash/manual/bash.html>

2.6 Hola Mundo

“Hola Mundo” es un ejemplo clásico que se utiliza para mostrar el funcionamiento básico de un lenguaje de programación.

Ejemplo:

En este ejemplo, se imprime el texto “Hola Mundo” en la consola.



3 GRAMÁTICA

También podrías modificar este código con tu nombre. Si es “Juan”, debería imprimir en la consola “Hola, Juan” .

Reto:

Modifica el ejemplo anterior para imprimir “Hola Universo” en la consola.

3 Gramática

3.1 Sintaxis

Bash es un lenguaje de scripting de Unix que se utiliza para automatizar tareas comunes. La sintaxis de Bash es relativamente sencilla y se basa en palabras clave, comandos, variables, parámetros y operadores. La indentación no es necesaria, pero se recomienda para mejorar la legibilidad. El conjunto de caracteres utilizado en Bash es ASCII, y la sensibilidad a mayúsculas y minúsculas es importante. Los comandos y variables deben escribirse exactamente como se definen para que Bash los reconozca.

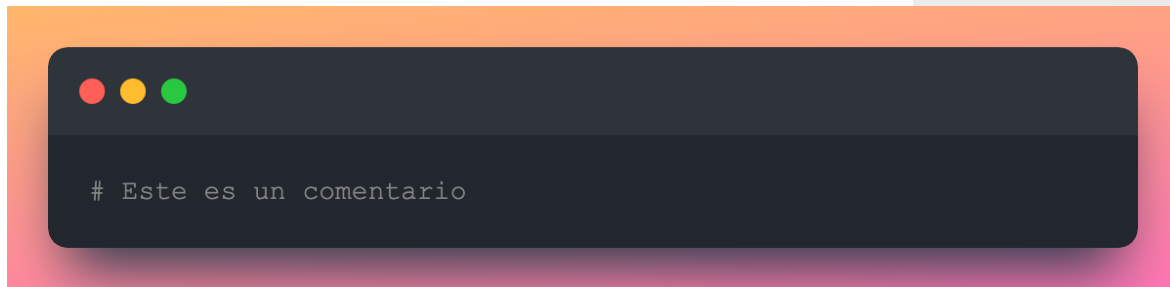
3.2 Comentarios

Los comentarios en Bash son líneas de texto que no son interpretadas como parte del código. Se usan para proporcionar información adicional sobre el código, como explicaciones, notas, o instrucciones para el desarrollador. Los comentarios también se pueden usar para deshabilitar código temporalmente, sin tener que eliminarlo completamente.

Ejemplo:

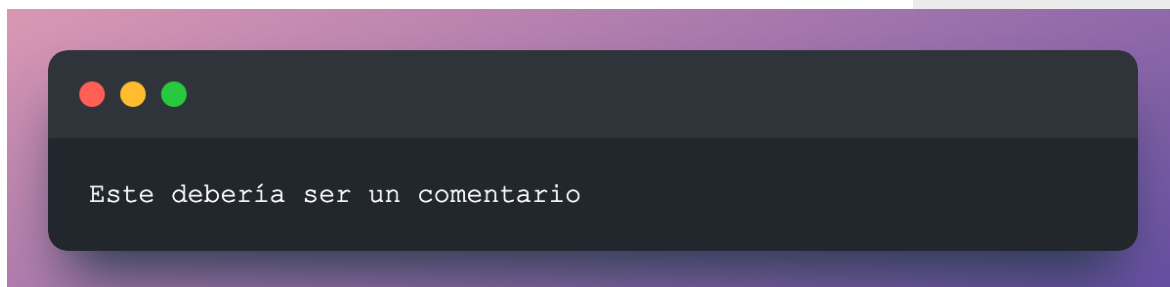
Este código es un comentario.

4 TIPOS Y VARIABLES



Reto:

Comenta esta línea para que no cause errores y se lea como comentario:



4 Tipos y variables

4.1 Variables

La manipulación de datos es una tarea fundamental de un lenguaje de programación. Para trabajar con datos necesitamos guardarlos en variables.


Una variable es un contenedor para almacenar datos que retiene su nombre y puede cambiar su valor a lo largo del tiempo. En los siguientes ejemplos vamos a ver varios tipos de datos que puedes guardar en variables.

Ejemplo:

Este código declara una variable llamada "libro" y le asigna el valor de una cadena

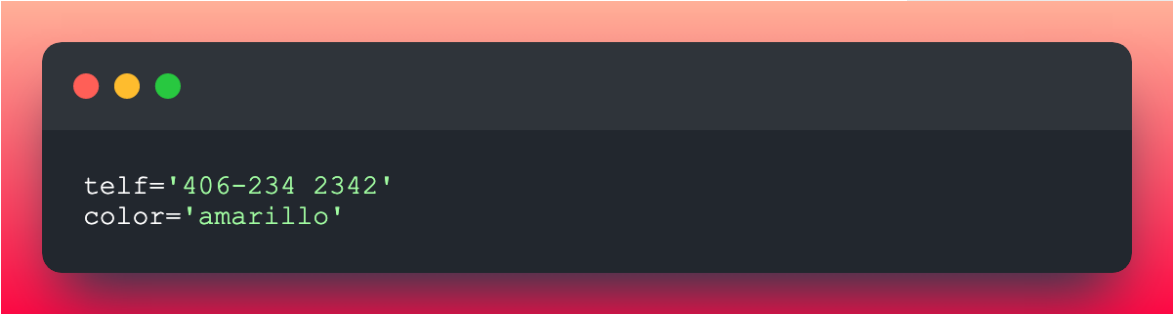
4 TIPOS Y VARIABLES

de texto que contiene el título de un libro. Luego, imprime el valor de la variable “libro” en la consola.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The terminal displays two lines of code: `libro="El Programador Pragmático"` and `echo $libro`.

```
libro="El Programador Pragmático"
echo $libro
```

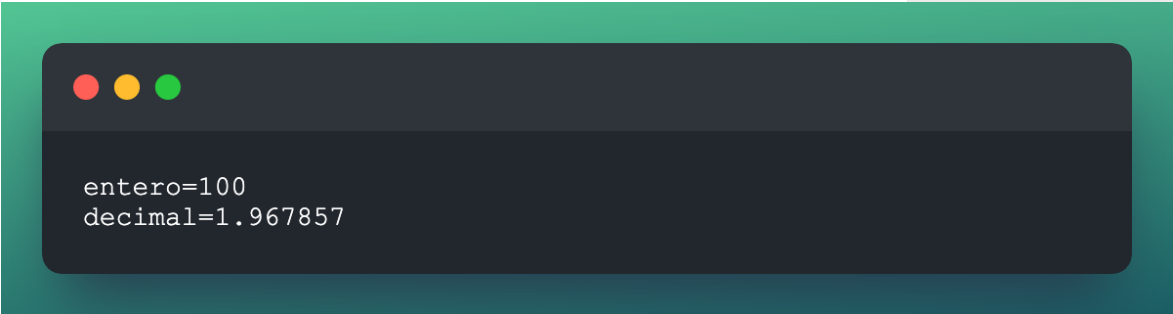
Texto es un tipo de dato útil para guardar información como números telefónicos y colores, entre otros. Este código asigna dos variables, una llamada telf que contiene un número de teléfono como una cadena de caracteres, y otra llamada color que contiene el color amarillo como una cadena de caracteres.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The terminal displays two lines of code: `telf='406-234 2342'` and `color='amarillo'`.

```
telf='406-234 2342'
color='amarillo'
```

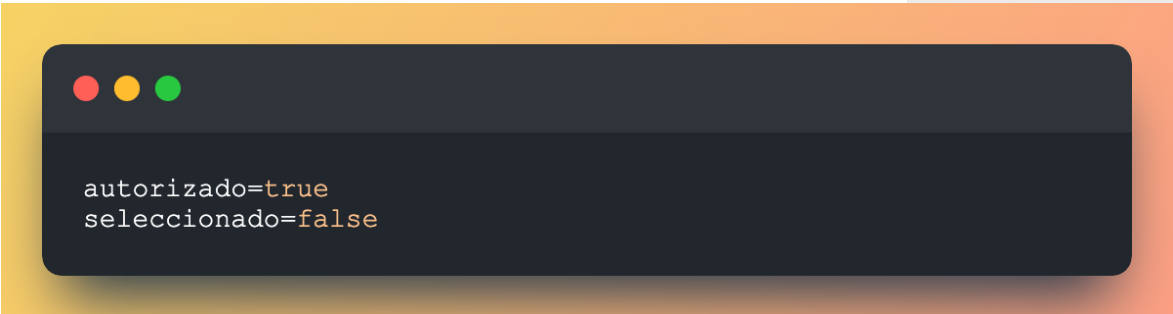
También podemos guardar datos como números enteros y decimales. Estos datos se usan para realizar operaciones matemáticas y representar valores de peso, dinero, entre otros. Este código asigna dos variables, una con un valor entero (100) y otra con un valor decimal (1.967857).

4 TIPOS Y VARIABLES



```
entero=100
decimal=1.967857
```

El tipo de dato booleano representa los valores de verdadero y falso. Este tipo de datos es útil, por ejemplo, para indicar si un usuario está autorizado a acceder a una app o no, entre varios usos. Este código crea una variable llamada “autorizado” que es verdadera y otra variable llamada “seleccionado” que es falsa.



```
autorizado=true
seleccionado=false
```

Es importante saber que, en el mundo del código binario, el número 1 representa verdadero y 0 representa falso.

Reto:

Crea una variable “a” con 33 como texto e imprímela a la consola.

4.2 Listas

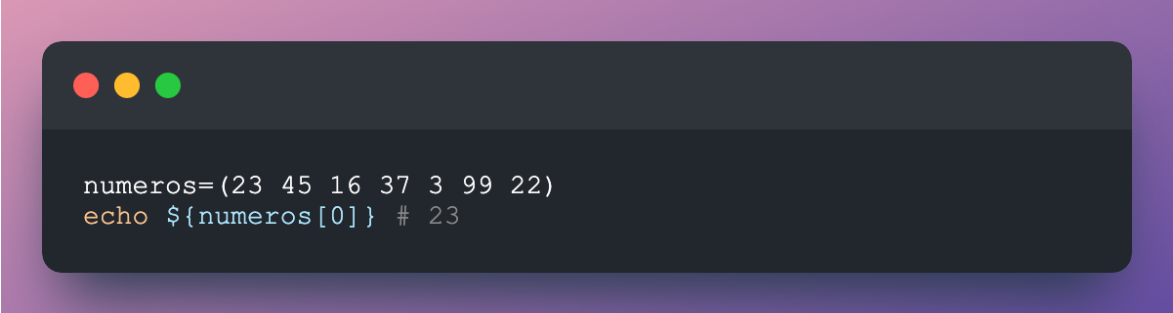
Las listas en Bash son una estructura de datos que nos permite almacenar una colección ordenada de elementos. Estos elementos pueden ser de cualquier tipo, desde números hasta sublistas. También los vas a encontrar con otros nombres

4 TIPOS Y VARIABLES

como arreglos, matrices, arrays, etc.

Ejemplo:

Este código está imprimiendo el primer elemento de una matriz de números a la consola. En este caso, el primer elemento es 23.



```
numeros=(23 45 16 37 3 99 22)
echo ${numeros[0]} # 23
```


También existen listas de texto. Este código crea una variable llamada “animales” que contiene una lista de elementos, en este caso, tres animales: perro, gato y tigre.



```
animales=("perro" "gato" "tigre")
```

Y esta es una lista de datos mixtos. Este código crea una variable llamada “datosMixtos” que contiene una lista de elementos de diferentes tipos, como texto, números, booleanos y otra lista.

4 TIPOS Y VARIABLES



```
datosMixtos=('texto' 69 true ['lista dentro de otra lista'])
```

Reto:

Crea una lista en Bash que contenga los nombres de los meses del año. Accede al 3er índice e imprímelo en la consola.

4.3 Objetos

Los objetos en Bash son una forma de almacenar y organizar datos mapeándolos de uno a uno. Estos datos se almacenan en forma de pares clave-valor, donde la clave suele ser una cadena de texto y el valor puede ser cualquier tipo de dato. También los vas a encontrar con otros nombres como mapas, diccionarios, etc.

Ejemplo:

Este código crea un objeto llamado jugadores que contiene dos pares clave-valor. El primer par clave-valor es 10: 'Messi' y el segundo es 7: 'Cristiano Ronaldo' . Luego, se imprime el valor asociado con la clave 10, que es 'Messi' .



```
jugadores=( [10]='Messi' [7]='Cristiano Ronaldo' )  
echo ${jugadores[10]} # 'Messi'
```

4 TIPOS Y VARIABLES

También podemos mapear de texto a texto. Este código crea un objeto llamado “países” que contiene claves y valores. Las claves son códigos de países (EC, MX, AR) y los valores son los nombres de los países (Ecuador, México, Argentina).

```
países=(  
  [EC]='Ecuador'  
  [MX]='México'  
  [AR]='Argentina'  
)
```

E incluso podemos mapear de texto a listas de texto, entre muchas otras opciones. Este código establece un objeto llamado “emails” que contiene dos pares clave-valor. Cada clave es un nombre de persona y el valor es un arreglo de direcciones de correo electrónico asociadas con esa persona.

```
emails=(  
  ["Juan"]="juan@gmail.com"  
  ["Ricardo"]="ricardo@gmail.com rick@aol.com"  
)
```

Reto:

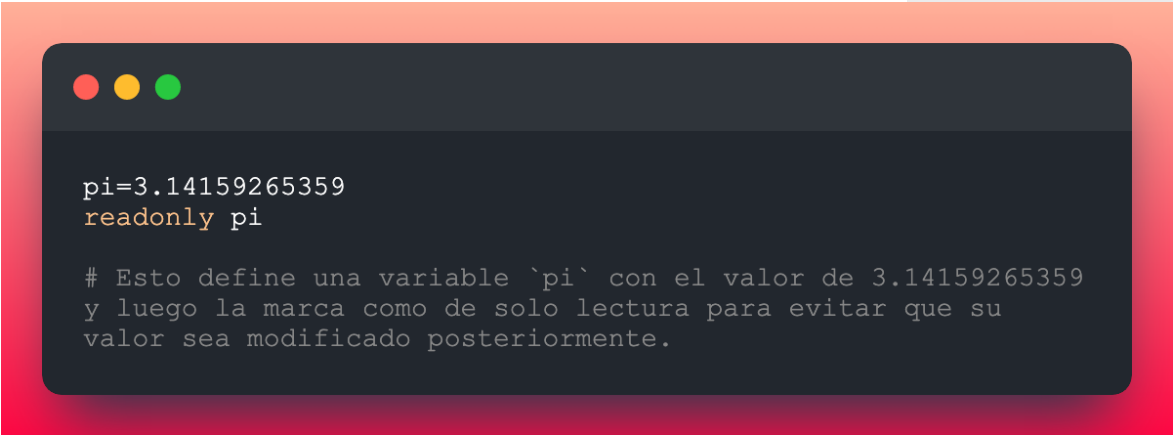
Crea un objeto en Bash que represente una computadora, con sus respectivas características (marca, modelo, procesador, memoria RAM, etc).

4.4 Constantes

Las constantes son variables que no pueden ser reasignadas. Esto significa que una vez que se asigna un valor a una constante, este no puede ser cambiado.

Ejemplo:

Esta línea de código establece una constante llamada “pi” con un valor de 3.14159265359. Esta constante se puede usar para almacenar un valor numérico que no cambiará a lo largo del programa.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The terminal displays the following text:

```
pi=3.14159265359
readonly pi

# Esto define una variable `pi` con el valor de 3.14159265359
y luego la marca como de solo lectura para evitar que su
valor sea modificado posteriormente.
```

Reto:

Crea una constante llamada ‘SALUDO’ y asígnale el valor ‘Hola Planeta’ . Luego, imprime el valor de la constante en la consola.

5 Operadores

5.1 Operadores de aritméticos

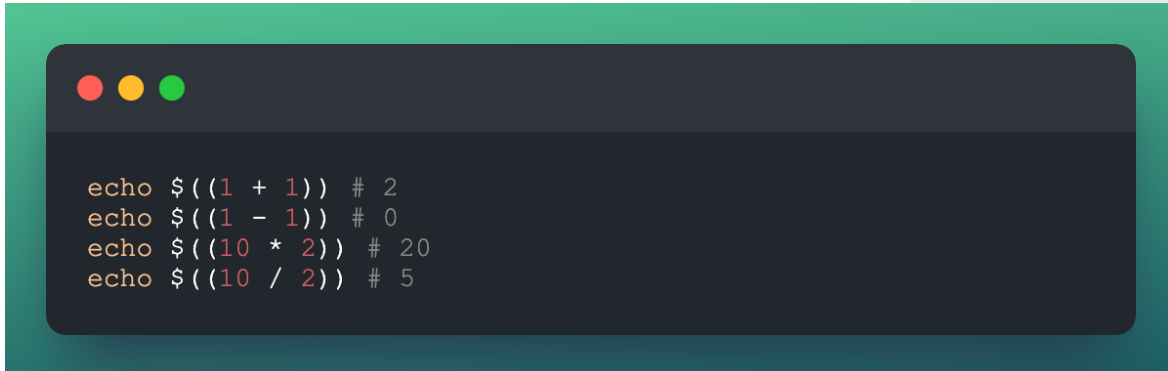
Los operadores aritméticos en Bash son usados para realizar operaciones matemáticas básicas. Estos operadores incluyen sumar, restar, multiplicar, dividir, entre

5 OPERADORES

otros.

Ejemplo:

Este código realiza operaciones matemáticas básicas, imprimiendo los resultados en la consola. Estas operaciones incluyen suma, resta, multiplicación y división.

A terminal window with a dark background and a green title bar. It displays four lines of Bash commands using the 'echo' command to show the results of arithmetic operations: addition, subtraction, multiplication, and division.

```
echo $((1 + 1)) # 2
echo $((1 - 1)) # 0
echo $((10 * 2)) # 20
echo $((10 / 2)) # 5
```

Reto:

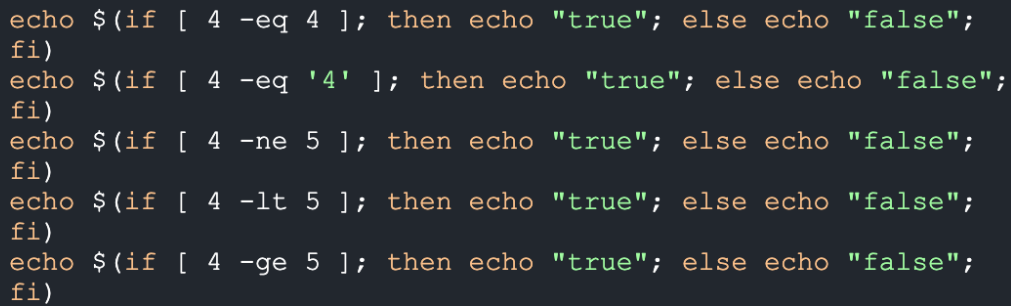
Usa los operadores aritméticos para calcular el área de un círculo con radio 5.

5.2 Operadores comparativos

Los operadores comparativos en Bash son usados para comparar dos valores y determinar si son iguales o diferentes. Estos operadores son útiles para determinar si dos valores son iguales, si dos valores no son iguales, si un valor es mayor o menor que otro, entre otros.

Ejemplo:

Este código muestra cómo se usan los operadores para comparar valores.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains six lines of Bash code using the 'if' statement with various comparison operators: -eq, -eq '4', -ne, -lt, and -ge. Each line is followed by 'then echo "true"; else echo "false"; fi)'.

```
echo $(if [ 4 -eq 4 ]; then echo "true"; else echo "false";  
fi)  
echo $(if [ 4 -eq '4' ]; then echo "true"; else echo "false";  
fi)  
echo $(if [ 4 -ne 5 ]; then echo "true"; else echo "false";  
fi)  
echo $(if [ 4 -lt 5 ]; then echo "true"; else echo "false";  
fi)  
echo $(if [ 4 -ge 5 ]; then echo "true"; else echo "false";  
fi)
```

Reto:

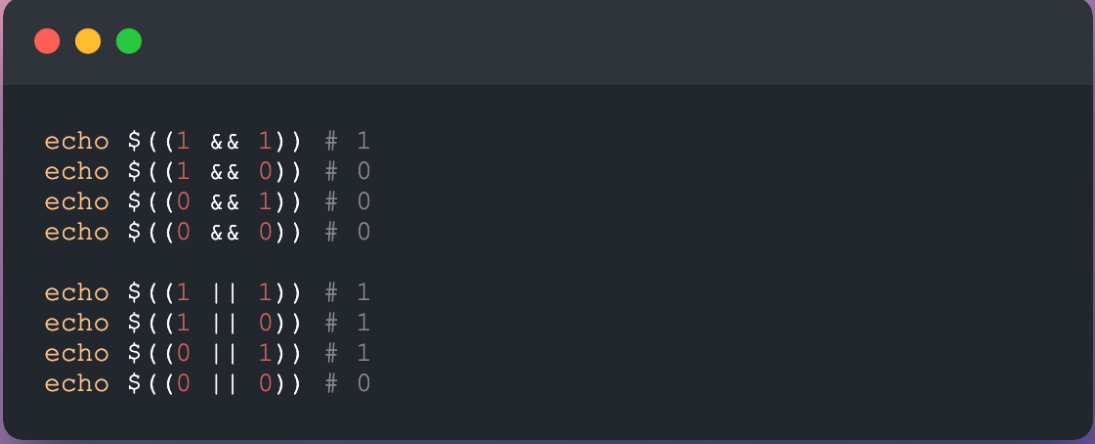
Escribe un programa que compare dos números 'a' y 'b' y determina si 'a' con el valor de 4 es mayor, menor o igual a 'b' con un valor de 2.

5.3 Operadores lógicos

Los operadores lógicos en Bash se usan para realizar comparaciones entre valores y devolver un valor booleano (verdadero o falso). Estos operadores puede ser útiles, por ejemplo, si deseamos saber si un 'animal' es un gato y es un mamífero (al mismo tiempo).

Ejemplo:

Este código muestra el uso de los operadores lógicos 'y' y 'o' para evaluar expresiones booleanas. El operador 'y' devuelve verdadero si ambas expresiones son verdaderas, de lo contrario devuelve falso. El operador 'o' devuelve verdadero si al menos una de las expresiones es verdadera, de lo contrario devuelve falso.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It displays two groups of bash conditional tests. The first group uses the '&&' operator for AND logic, and the second group uses the '||' operator for OR logic. Each line shows the command 'echo' followed by a test expression and a comment indicating the expected result.

```
echo $((1 && 1)) # 1
echo $((1 && 0)) # 0
echo $((0 && 1)) # 0
echo $((0 && 0)) # 0

echo $((1 || 1)) # 1
echo $((1 || 0)) # 1
echo $((0 || 1)) # 1
echo $((0 || 0)) # 0
```

Reto:

Escribe una línea de código que devuelva verdadero si 'x' es mayor que 0 y 'y' es menor que 0.

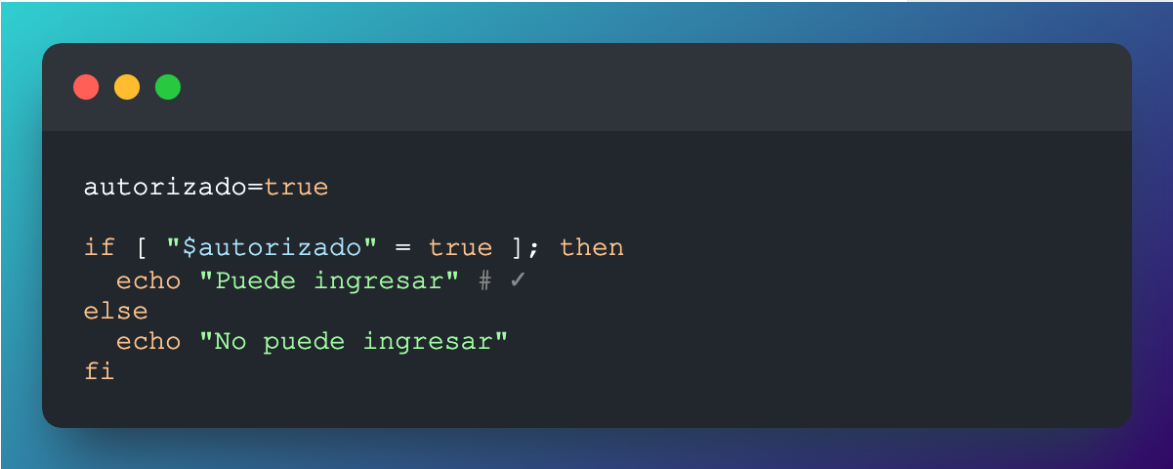
6 Condicionales

6.1 Condición única

Los condicionales son estructura de control de flujo en Bash, es decir, controlan el flujo del código. Permiten ejecutar una sección de código si una condición es verdadera. También permiten ejecutar otra sección de código, si la condición es falsa.

Ejemplo:

Este código comprueba si una variable booleana (autorizado) es verdadera. Si es así, imprime un mensaje en la consola indicando que el usuario puede ingresar. Si no es así, imprime un mensaje indicando que el usuario no puede ingresar.

A terminal window with a dark background and a blue header bar. It contains a shell script snippet. The script starts with 'autorizado=true', followed by an 'if' statement checking if '\$autorizado' equals 'true'. If true, it echoes 'Puede ingresar' with a checkmark. Otherwise, it echoes 'No puede ingresar'. The script ends with 'fi'.

```
autorizado=true

if [ "$autorizado" = true ]; then
    echo "Puede ingresar" # ✓
else
    echo "No puede ingresar"
fi
```

Reto:

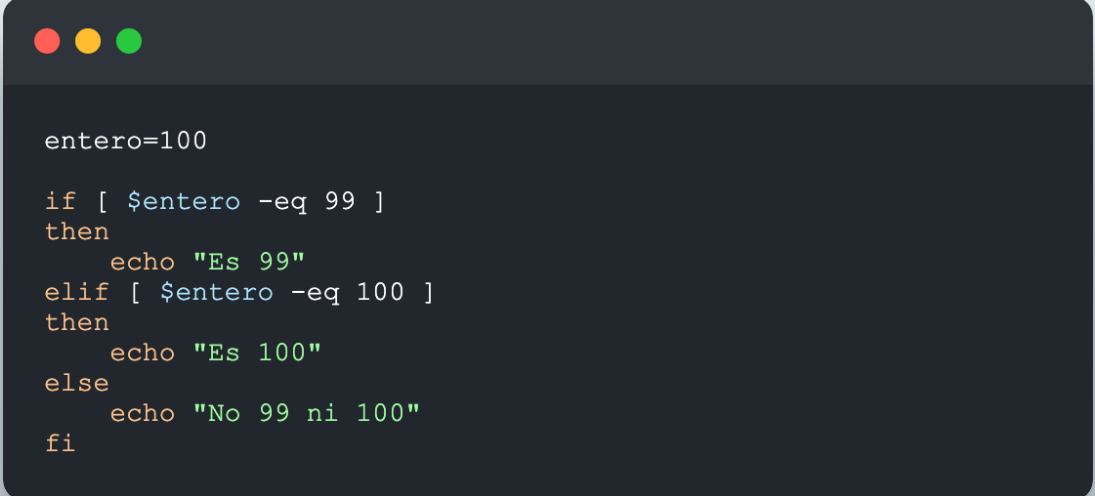
Escribe código condicional que revise si un número 'a' es par o impar. La variable 'a' tiene el valor de 17 y debe imprimir a la consola 'Es par' si es par o 'Es impar' si es impar.

6.2 Múltiples condiciones

Adicionalmente, los condicionales permiten ejecutar varias secciones de código de acuerdo a varias condiciones.

Ejemplo:

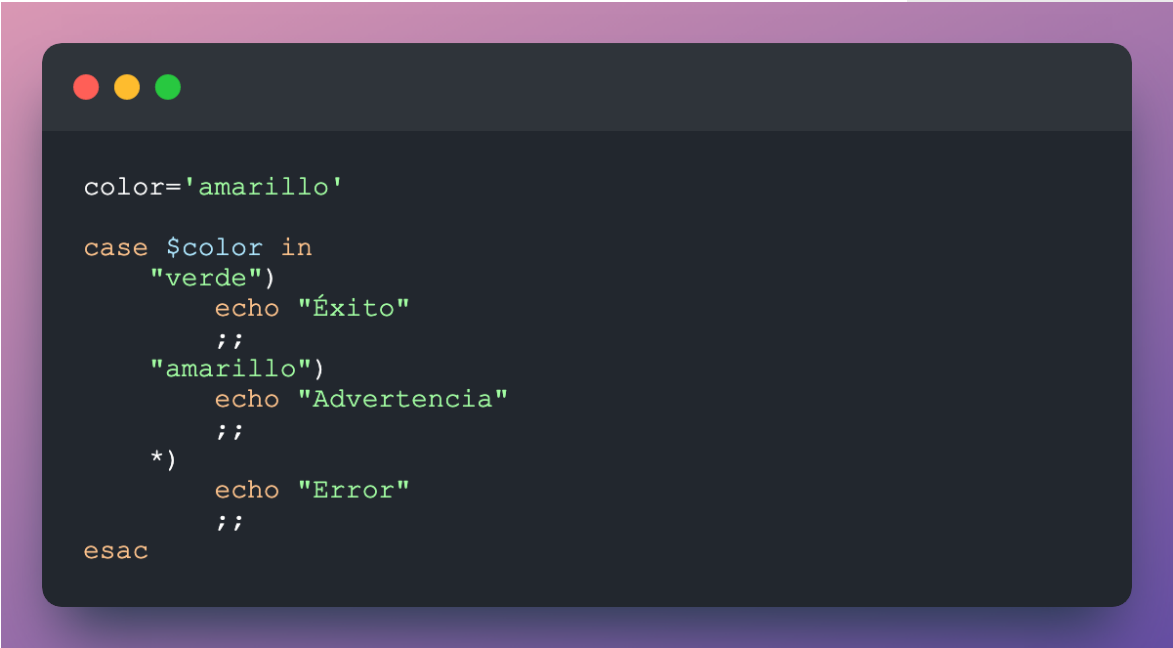
En este caso podemos ver que el código que se ejecuta depende de varios valores. Este código comprueba si una variable llamada entero es igual a 99 o 100. Si es igual a 99, imprime "Es 99" en la consola. Si es igual a 100, imprime "Es 100" en la consola. Si no es ni 99 ni 100, imprime "No 99 ni 100" en la consola.



```
entero=100

if [ $entero -eq 99 ]
then
    echo "Es 99"
elif [ $entero -eq 100 ]
then
    echo "Es 100"
else
    echo "No 99 ni 100"
fi
```

Este condicional es útil cuando hay una gran cantidad de posibles resultados para una expresión. Este código compara una variable (color) con diferentes valores y ejecuta una acción dependiendo del resultado de la comparación. En este caso, si la variable color es igual a 'amarillo', se imprimirá 'Advertencia' en la consola.



```
color='amarillo'

case $color in
    "verde")
        echo "Éxito"
        ;;
    "amarillo")
        echo "Advertencia"
        ;;
    *)
        echo "Error"
        ;;
esac
```

Reto:

Crea un programa que evalúe una variable 'numero' y dependiendo del resultado, imprima un mensaje diferente. Si el número es mayor a 10, imprime "El número es mayor a 10" . Si el número es menor a 10, imprime "El número es menor a 10" . Si el número es igual a 10, imprime "El número es igual a 10" .

7 Bucles


7.1 Bucle for

Los bucles son otra estructura de control de flujo que se utilizan para iterar sobre una secuencia de elementos. También se los llama loops o ciclos.

Ejemplo:

7 BUCLES

Este código itera sobre una lista de animales e imprime cada elemento de la lista en la consola.



```
animales=("perro" "gato" "tigre")

for animal in "${animales[@]}"
do
    echo "$animal"
done

# 'perro'
# 'gato'
# 'tigre'
```

Reto:

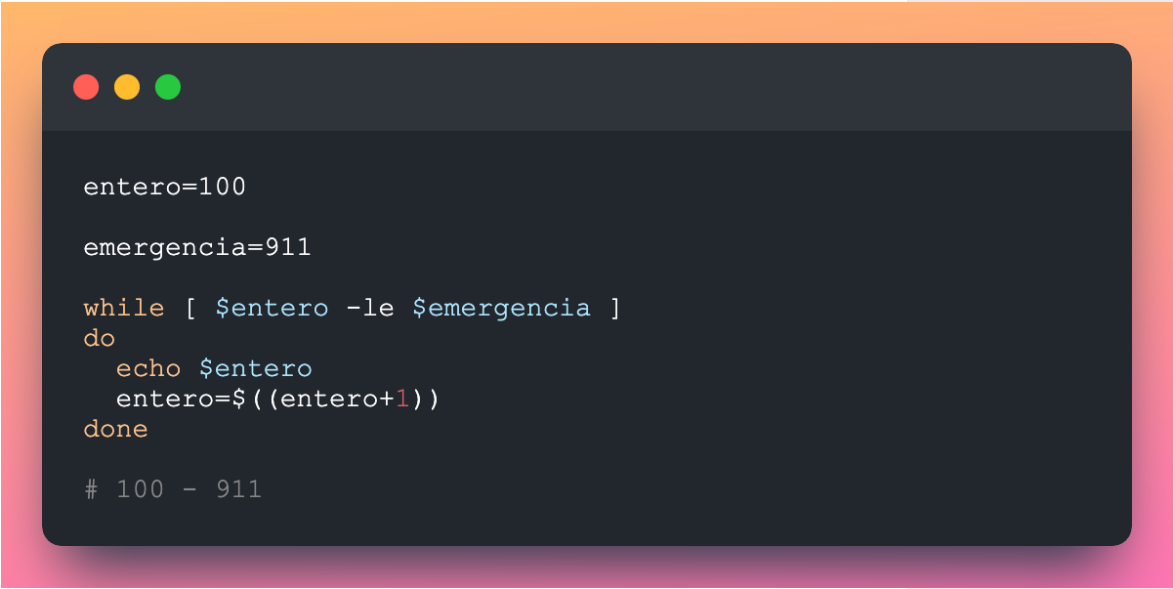
Escribe un bucle que imprima los números del 1 al 10 en orden ascendente.

7.2 Bucle while

while es un tipo de bucle en Bash que se usa para ejecutar una serie de instrucciones mientras se cumpla una condición. Esta condición se evalúa antes de cada iteración del bucle.

Ejemplo:

Este código establece un bucle while que imprime los números enteros desde 100 hasta 911 en la consola. El bucle while se ejecutará mientras la variable entero sea menor o igual a la variable emergencia. Cada vez que el bucle se ejecuta, la variable entero se incrementa en 1.



```
entero=100

emergencia=911

while [ $entero -le $emergencia ]
do
    echo $entero
    entero=$((entero+1))
done

# 100 - 911
```

Ten cuidado de no incluir una condición para parar el bucle. Esto podría seguir corriendo tu código indefinidamente hasta congelar tu computador.

Reto:

Crea un bucle while que imprima los números del 1 al 10 en la consola.

8 Funciones

En Bash, una función es un bloque de código diseñado para realizar una tarea específica y se puede reutilizar en varias partes el código. Las funciones se pueden definir para realizar cualquier tarea, desde realizar cálculos hasta mostrar mensajes en la pantalla.

Ejemplo:

Esta función toma dos argumentos (primero y segundo) de forma dinámica y los suma. Luego, imprime el resultado en la consola. Esta función se llama dos veces con diferentes argumentos para mostrar los resultados de la suma.

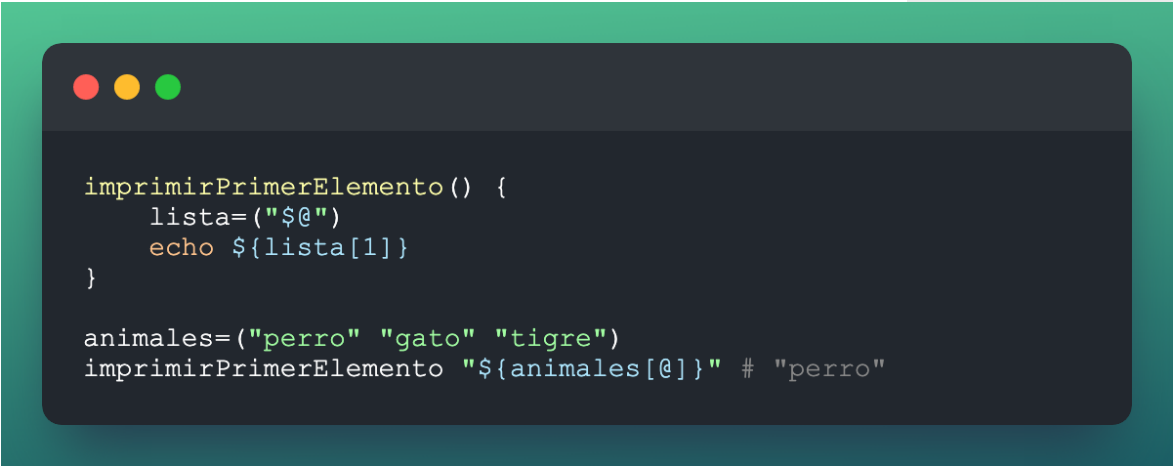
8 FUNCIONES

```
function sumar {  
    echo $(( $1 + $2 ))  
}  
  
sumar 2 2 # 4  
sumar 3 4 # 7
```

También puedes crear funciones que retornen un valor. Esto significa que una vez que se ejecuta una función, el valor devuelto se puede usar en otra parte del código. Esta función toma dos argumentos (primero y segundo) y los multiplica para devolver el resultado. En este caso, el resultado es 6.

```
multiplicar() {  
    echo $(( $1 * $2 ))  
}  
  
resultado=$(multiplicar 3 2)  
echo $resultado # 6
```

Esta función imprime el primer elemento de una lista. En este caso, la lista es una lista de animales, y la función imprime el primer elemento de la lista, que es 'perro'. Funciones como esta son útiles para evitar la repetición de código y para ahorrar tiempo.



```
imprimirPrimerElemento() {  
    lista=("$@")  
    echo ${lista[1]}  
}  
  
animales=("perro" "gato" "tigre")  
imprimirPrimerElemento "${animales[@]}" # "perro"
```

Finalmente, puedes ver otro ejemplo de una función bastante compleja. Esta función implementa el algoritmo de ordenamiento QuickSort para ordenar una lista de elementos. El algoritmo comienza tomando un elemento de la lista como pivote y luego coloca los elementos menores que el pivote a su izquierda y los elementos mayores a su derecha. Luego, se aplica el mismo algoritmo a las sublistas izquierda y derecha hasta que la lista esté ordenada. No tienes que entender todo lo que hace internamente pero te dará una idea de la complejidad que pueden tener programas como apps con miles de funciones.

```
quicksort() {
    lista=("$@")
    if (( ${#lista[@]} <= 1 )); then
        echo "${lista[@]}"
        return
    fi
    pivote=${lista[0]}
    izquierda=()
    derecha=()
    for (( i=1; i<${#lista[@]}; i++ )); do
        if (( ${lista[i]} < pivote )); then
            izquierda+=("${lista[i]}")
        else
            derecha+=("${lista[i]}")
        fi
    done
    echo $(quicksort "${izquierda[@]}") $pivote $(quicksort
"${derecha[@]}")
}

numeros=(23 45 16 37 3 99 22)
listaOrdenada=$(quicksort "${numeros[@]}")
echo "${listaOrdenada[@]}" # 3 16 22 23 37 45 99
```

Reto:

Escribe una función que tome dos números como argumentos y devuelva el mayor de los dos.

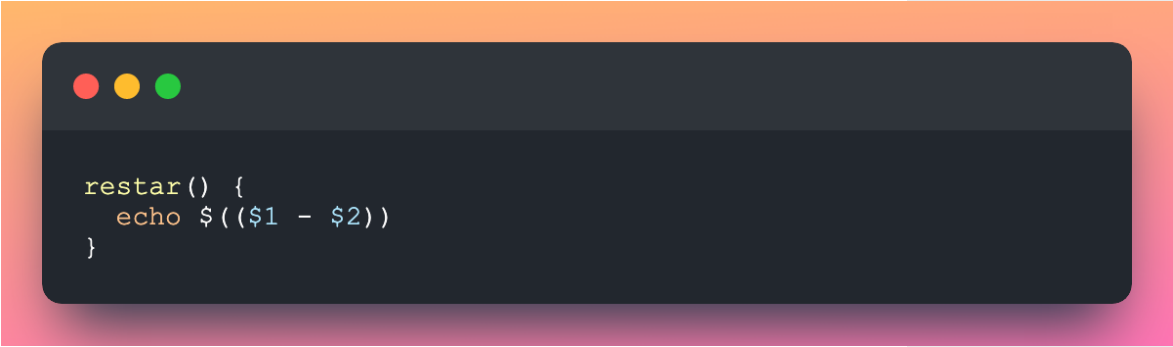
9 Módulos

Los módulos son una forma de incluir código externo de otro archivo. Esto permite a los desarrolladores reutilizar código y ahorrar tiempo al escribir código.

9 MÓDULOS

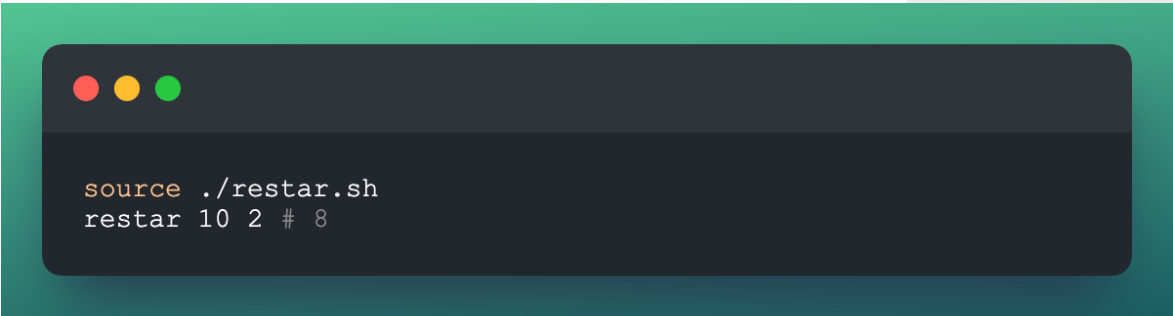
Ejemplo:

Este archivo define una función que toma dos argumentos (a y b) y luego imprime el resultado de la resta de a y b en la consola. El código exporta la función para que pueda ser usada en otros archivos. Exportar es una forma de compartir código entre archivos. Esto significa que un archivo puede exportar variables, funciones y objetos para que otros archivos los puedan usar.



```
restar() {  
  echo $(( $1 - $2 ))  
}
```

Y en otro archivo podemos importar y utilizar esta función. Este código importa una función desde el archivo externo y luego la usa para restar 10 menos 2, resultando en 8.



```
source ./restar.sh  
restar 10 2 # 8
```

Reto:

Crea un archivo llamado que contenga una función para calcular el área de un rectángulo. Luego, importa la función en otro archivo y usala para calcular el área de un rectángulo con lados de 3 y 4.

10 Siguientes pasos

10.1 Herramientas

1. **Git:** Git es una herramienta de control de versiones de código abierto que permite a los desarrolladores rastrear y administrar cambios en sus proyectos. Esto permite a los desarrolladores trabajar en proyectos de forma colaborativa y mantener un historial de cambios.

Enlace: <https://git-scm.com/>

2. **Vim:** Vim es un editor de texto de código abierto para Linux y Unix. Está diseñado para ser una herramienta de edición de texto rápida y eficiente. Vim es una herramienta muy útil para los desarrolladores que trabajan con Bash.

Enlace: <https://www.vim.org/>

3. **Grep:** Grep es una herramienta de línea de comandos que se utiliza para buscar y filtrar cadenas de texto en archivos de texto. Esta herramienta es muy útil para los desarrolladores que trabajan con Bash, ya que les permite buscar y filtrar cadenas de texto en archivos de texto.

Enlace: <https://www.gnu.org/software/grep/>

4. **Awk:** Awk es un lenguaje de programación de alto nivel diseñado para procesar archivos de texto. Esta herramienta es muy útil para los desarrolladores que trabajan con Bash, ya que les permite procesar archivos de texto de forma rápida y eficiente.

Enlace: <https://www.gnu.org/software/gawk/>

10.2 Recursos

1. [The Linux Documentation Project](#): El Proyecto de Documentación de Linux

es una colección de documentos sobre el uso de Linux. Está diseñado para ayudar a los usuarios a aprender cómo usar y administrar sistemas basados en Linux.

2. [BashGuide](#): BashGuide es una guía completa para el uso de Bash. Está diseñado para ayudar a los usuarios a aprender cómo usar Bash de manera eficiente y eficaz.
3. [Bash-hackers Wiki](#): Bash-hackers Wiki es una colección de recursos para el uso de Bash. Está diseñado para ayudar a los usuarios a aprender cómo usar Bash de manera eficiente y eficaz.
4. [Bash Reference Manual](#): El Manual de Referencia de Bash es una guía completa para el uso de Bash. Está diseñado para ayudar a los usuarios a aprender cómo usar Bash de manera eficiente y eficaz.
5. [Bash Cookbook](#): Bash Cookbook es una colección de recetas para el uso de Bash. Está diseñado para ayudar a los usuarios a aprender cómo usar Bash de manera eficiente y eficaz.

10.3 ¿Que viene después?

1. Aprender a usar herramientas de línea de comandos como grep, sed y awk.
2. Aprender a escribir scripts Bash para automatizar tareas.
3. Aprender a administrar procesos y servicios en el sistema operativo.
4. Aprender a administrar usuarios y grupos.
5. Aprender a administrar archivos y directorios.
6. Aprender a administrar paquetes y repositorios.
7. Aprender a administrar la red y los servicios de red.
8. Aprender a administrar la seguridad del sistema.

10.4 Preguntas de entrevista

1. ¿Qué es Bash?

- Bash es un intérprete de línea de comandos para el sistema operativo Unix.

2. ¿Qué significa la palabra Bash?

- Bash significa Bourne-Again Shell.

3. ¿Qué es un shell?

- Un shell es un intérprete de línea de comandos que proporciona una interfaz entre el usuario y el sistema operativo.

4. ¿Qué es un intérprete de línea de comandos?

- Un intérprete de línea de comandos es un programa que lee y ejecuta órdenes escritas en un lenguaje de programación específico.

5. ¿Qué es una variable de entorno?

- Una variable de entorno es una variable que se utiliza para almacenar información sobre el entorno de un usuario, como la ubicación de los archivos, el directorio de trabajo actual, etc.

6. ¿Cómo se crea una variable de entorno?

- Se crea una variable de entorno utilizando el comando export seguido del nombre de la variable y el valor que desea asignar.

7. ¿Cómo se puede ver el contenido de una variable de entorno?

- Se puede ver el contenido de una variable de entorno utilizando el comando echo seguido del nombre de la variable.

8. ¿Qué es una función?

10 SIGUIENTES PASOS

- Una función es un bloque de código que se puede reutilizar para realizar una tarea específica.
9. ¿Cómo se define una función en Bash?
- Se define una función en Bash utilizando el comando `function` seguido del nombre de la función y el código que desea ejecutar.
10. ¿Qué es un bucle?
- Un bucle es una estructura de control que se utiliza para ejecutar un bloque de código repetidamente hasta que se cumpla una condición específica.
11. ¿Cuáles son los tipos de bucles en Bash?
- Los tipos de bucles en Bash son `for`, `while`, `until` y `select`.
12. ¿Cómo se crea un bucle `for` en Bash?
- Se crea un bucle `for` en Bash utilizando el comando `for` seguido de la variable que se desea iterar, el valor inicial y el valor final.
13. ¿Cómo se crea un bucle `while` en Bash?
- Se crea un bucle `while` en Bash utilizando el comando `while` seguido de la condición que se desea evaluar.
14. ¿Cómo se crea un bucle `until` en Bash?
- Se crea un bucle `until` en Bash utilizando el comando `until` seguido de la condición que se desea evaluar.
15. ¿Cómo se crea un bucle `select` en Bash?
- Se crea un bucle `select` en Bash utilizando el comando `select` seguido de la lista de opciones que se desea mostrar al usuario.
16. ¿Qué es una condición?

10 SIGUIENTES PASOS

- Una condición es una expresión booleana que se evalúa como verdadera o falsa.
17. ¿Cómo se evalúa una condición en Bash?
- Se evalúa una condición en Bash utilizando el comando if seguido de la condición que se desea evaluar.
18. ¿Qué son los comandos de redirección?
- Los comandos de redirección son comandos que se utilizan para redirigir la salida de un comando a un archivo o a la entrada de otro comando.
19. ¿Cuáles son los comandos de redirección más comunes?
- Los comandos de redirección más comunes son `>`, `>>`, `<` y `|`.
20. ¿Cómo se utiliza el comando `>`?
- El comando `>` se utiliza para redirigir la salida de un comando a un archivo.
21. ¿Cómo se utiliza el comando `>>`?
- El comando `>>` se utiliza para redirigir la salida de un comando al final de un archivo existente.
22. ¿Cómo se utiliza el comando `<`?
- El comando `<` se utiliza para redirigir la entrada de un comando desde un archivo.
23. ¿Cómo se utiliza el comando `|`?
- El comando `|` se utiliza para redirigir la salida de un comando a la entrada de otro comando.
24. ¿Qué son los comandos de control de flujo?
- Los comandos de control de flujo son comandos que se utilizan para controlar el flujo de ejecución de un programa.

25. ¿Cuáles son los comandos de control de flujo más comunes?

- Los comandos de control de flujo más comunes son break, continue y return.

26. ¿Cómo se utiliza el comando break?

- El comando break se utiliza para interrumpir un bucle.

27. ¿Cómo se utiliza el comando continue?

- El comando continue se utiliza para saltar a la siguiente iteración de un bucle.

28. ¿Cómo se utiliza el comando return?

- El comando return se utiliza para finalizar una función y devolver un valor.

29. ¿Qué son los comandos de procesamiento de texto?

- Los comandos de procesamiento de texto son comandos que se utilizan para manipular archivos de texto.

30. ¿Cuáles son los comandos de procesamiento de texto más comunes?

- Los comandos de procesamiento de texto más comunes son grep, sed y awk.

31. ¿Cómo se utiliza el comando grep?

- El comando grep se utiliza para buscar una cadena de texto en un archivo.

32. ¿Cómo se utiliza el comando sed?

- El comando sed se utiliza para editar un archivo de texto.

33. ¿Cómo se utiliza el comando awk?

- El comando awk se utiliza para procesar archivos de texto.

34. ¿Qué son los comandos de administración de procesos?

10 SIGUIENTES PASOS

- Los comandos de administración de procesos son comandos que se utilizan para administrar procesos en un sistema operativo.
35. ¿Cuáles son los comandos de administración de procesos más comunes?
- Los comandos de administración de procesos más comunes son ps, kill y top.
36. ¿Cómo se utiliza el comando ps?
- El comando ps se utiliza para mostrar información sobre los procesos en ejecución.
37. ¿Cómo se utiliza el comando kill?
- El comando kill se utiliza para terminar un proceso.
38. ¿Cómo se utiliza el comando top?
- El comando top se utiliza para mostrar una lista de los procesos más activos en un sistema.
39. ¿Qué son los comandos de administración de archivos?
- Los comandos de administración de archivos son comandos que se utilizan para administrar archivos y directorios en un sistema operativo.
40. ¿Cuáles son los comandos de administración de archivos más comunes?
- Los comandos de administración de archivos más comunes son ls, cp, mv y rm.
41. ¿Cómo se utiliza el comando ls?
- El comando ls se utiliza para mostrar el contenido de un directorio.
42. ¿Cómo se utiliza el comando cp?
- El comando cp se utiliza para copiar archivos y directorios.

43. ¿Cómo se utiliza el comando mv?

- El comando mv se utiliza para mover archivos y directorios.

44. ¿Cómo se utiliza el comando rm?

- El comando rm se utiliza para eliminar archivos y directorios.

45. ¿Qué son los comandos de administración de usuarios?

- Los comandos de administración de usuarios son comandos que se utilizan para administrar usuarios y grupos en un sistema operativo.

46. ¿Cuáles son los comandos de administración de usuarios más comunes?

- Los comandos de administración de usuarios más comunes son useradd, usermod, userdel y groupadd.

47. ¿Cómo se utiliza el comando useradd?

- El comando useradd se utiliza para agregar un usuario a un sistema.

48. ¿Cómo se utiliza el comando usermod?

- El comando usermod se utiliza para modificar la información de un usuario.

49. ¿Cómo se utiliza el comando userdel?

- El comando userdel se utiliza para eliminar un usuario de un sistema.

50. ¿Cómo se utiliza el comando groupadd?

- El comando groupadd se utiliza para agregar un grupo a un sistema.

51. ¿Qué son los comandos de administración de paquetes?

- Los comandos de administración de paquetes son comandos que se utilizan para administrar paquetes en un sistema operativo.

52. ¿Cuáles son los comandos de administración de paquetes más comunes?

10 SIGUIENTES PASOS

- Los comandos de administración de paquetes más comunes son apt, yum y rpm.
53. ¿Cómo se utiliza el comando apt?
- El comando apt se utiliza para administrar paquetes en sistemas basados en Debian.
54. ¿Cómo se utiliza el comando yum?
- El comando yum se utiliza para administrar paquetes en sistemas basados en Red Hat.
55. ¿Cómo se utiliza el comando rpm?
- El comando rpm se utiliza para administrar paquetes en sistemas basados en Red Hat.
56. ¿Qué son los comandos de administración de redes?
- Los comandos de administración de redes son comandos que se utilizan para administrar redes en un sistema operativo.
57. ¿Cuáles son los comandos de administración de redes más comunes?
- Los comandos de administración de redes más comunes son ifconfig, route y ping.
58. ¿Cómo se utiliza el comando ifconfig?
- El comando ifconfig se utiliza para mostrar y configurar la información de la interfaz de red.
59. ¿Cómo se utiliza el comando route?
- El comando route se utiliza para mostrar y configurar la tabla de enrutamiento.
60. ¿Cómo se utiliza el comando ping?

10 SIGUIENTES PASOS

- El comando ping se utiliza para probar la conectividad entre dos hosts.

61. ¿Qué son los comandos de depuración?

- Los comandos de depuración son comandos que se utilizan para depurar programas en un sistema operativo.