Computación en Física

Harvey Rodriguez Gil

Universidad EIA

8 de Agosto de 2024

Ciclos - for

El intérprete nos permite realizar una iteración sobre valores. Por medio del comando `for` podemos hacer esto. La sintaxis es la siguiente:

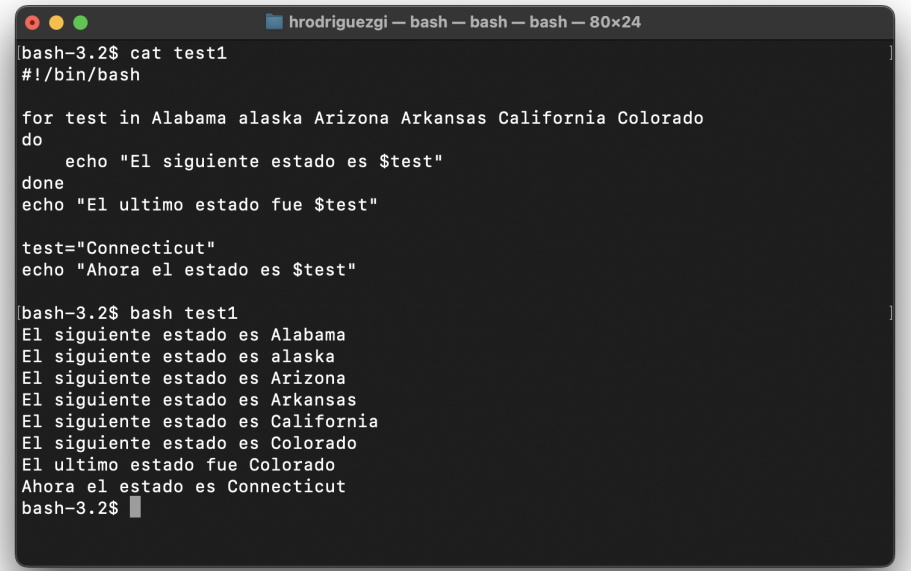
```
for var in list
do
 comandos
done
```

Esto también se puede escribir en una sola linea de código:

```
for var in list; do comandos; done
```

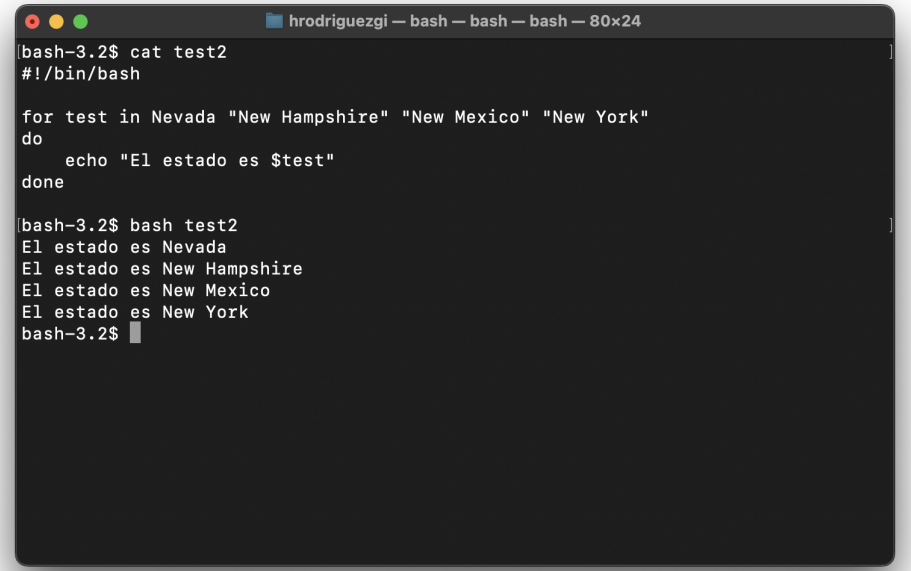
Ciclos - for

Una primera alternativa es definir los valores que se quieren iterar posterior de la palabra `in` del comando `for`:



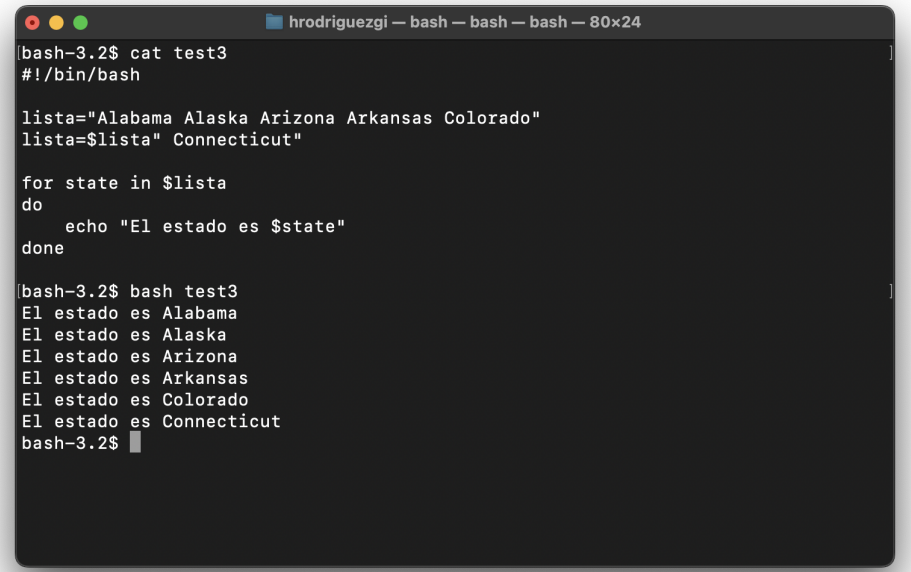
Ciclos - for

Cuando se cuentan con elementos en la lista que tienen un espacio como lo es el caso de **New York** en el ejemplo, se deben colocar los elementos entre comillas (")



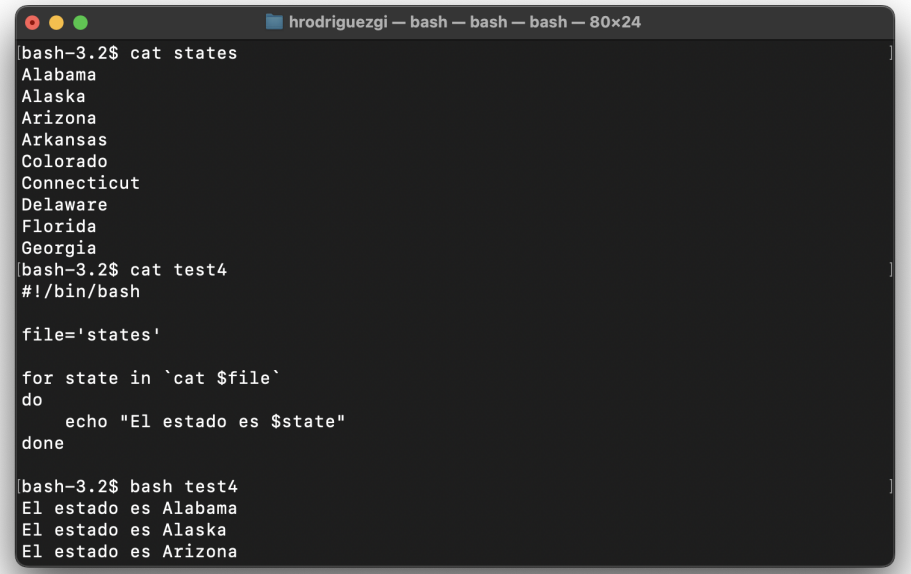
Ciclos - for

También es posible adicionar elementos a una variable, ya sea con un valor sencillo como en el ejemplo o por la concatenación de dos o más variables:



Ciclos - for

Con ayuda de los backticks incluso podemos extender la funcionalidad de `for` por medio de algún comando que se ejecute y nos devuelva un objeto que se pueda iterar:



Ciclos - for

Los ciclos `for` pueden controlarse por medio de dos comandos `break` que le indica a bash salir del ciclo aun cuando no se haya terminado (cumplimiento de alguna condición)

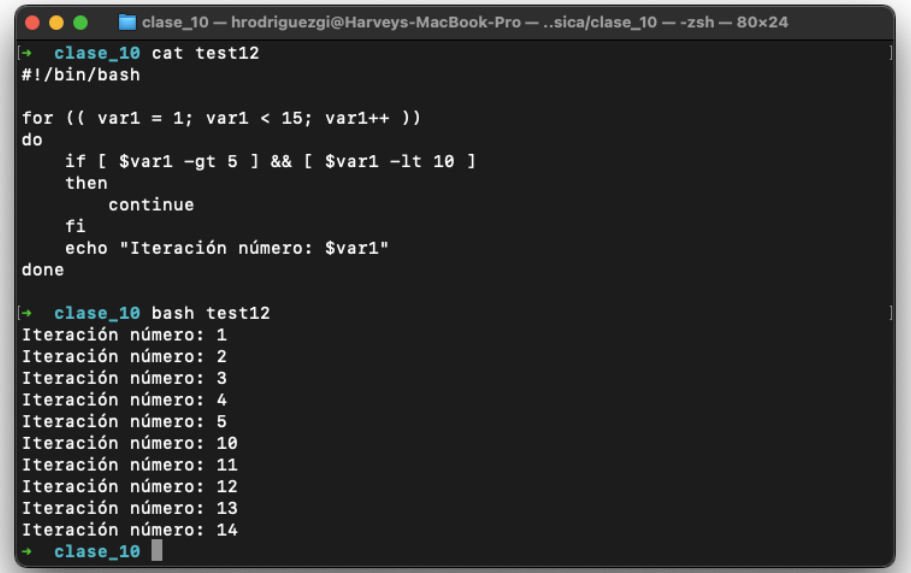
clase_10 — hrodriguezgi@Harveys-MacBook-Pro — ..sica/clase_10 — -zsh — 80×24

```
→  clase_10 cat test10
#!/bin/bash

for var1 in {1..10}
do
    if [ $var1 -eq 5 ]
    then
        break
    fi
    echo "Valor de la variable: $var1"
done
echo "El ciclo se ha completado"

→  clase_10 bash test10
Valor de la variable: 1
Valor de la variable: 2
Valor de la variable: 3
Valor de la variable: 4
El ciclo se ha completado
→  clase_10
```

UNIVERSIDAD EIA
Ser, Saber y Servir

Ciclos - for

Por otro lado con el comando `continue` se le indica que continúe con el siguiente elemento de la iteración



Ciclos - while

A diferencia del `for`, el comando `while` no itera sobre una lista, si no que depende del cumplimiento de una condición para ejecutar un comando repetidamente. La sintaxis es la siguiente:

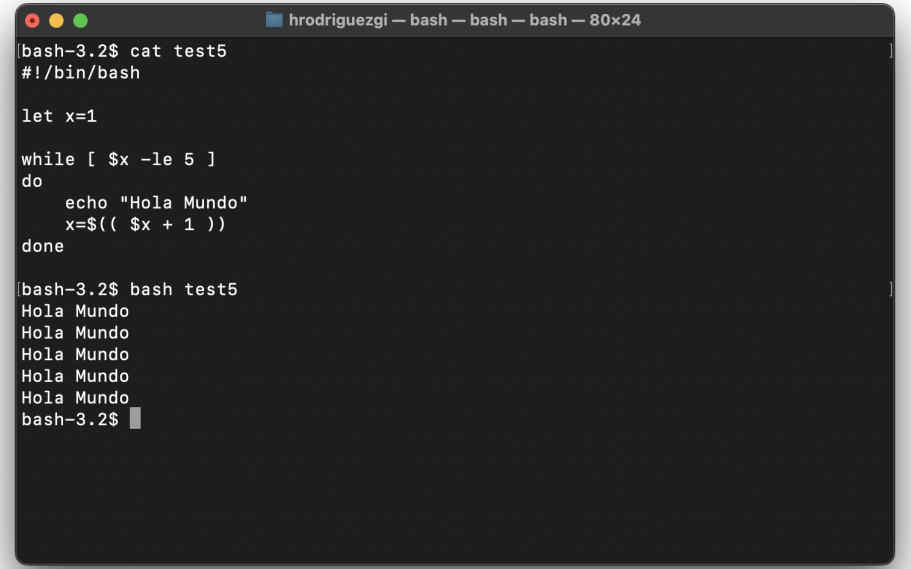
```
while condición
do
 comandos
done
```

Esto también se puede escribir en una sola linea de código:

```
while condición; do comandos; done
```

Ciclos - while

Es importante tener en cuenta en los ciclos `while` de incrementar nuestra variable (si este es nuestro condicional), de otro modo se quedará infinitamente en el ciclo:

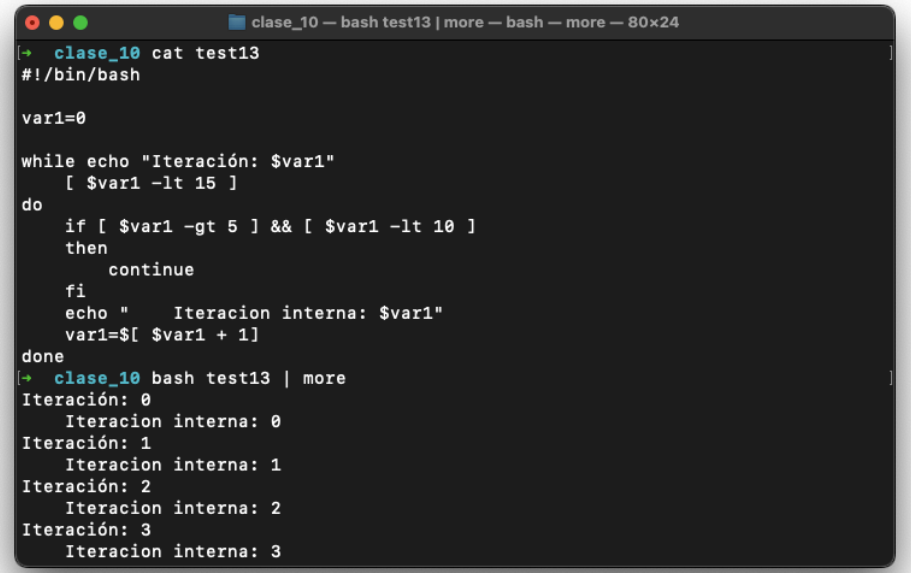


Ciclos - while

Los ciclos `while` también pueden controlarse por medio de dos comandos `break` que le indica a bash salir del ciclo aun cuando no se haya terminado (cumplimiento de alguna condición)

Ciclos - while

Por otro lado con el comando `continue` se le indica que continúe con el siguiente elemento de la iteración



Funciones

Como en los lenguajes de programación, las funciones nos permiten crear bloques de códigos que pueden ser reutilizados.

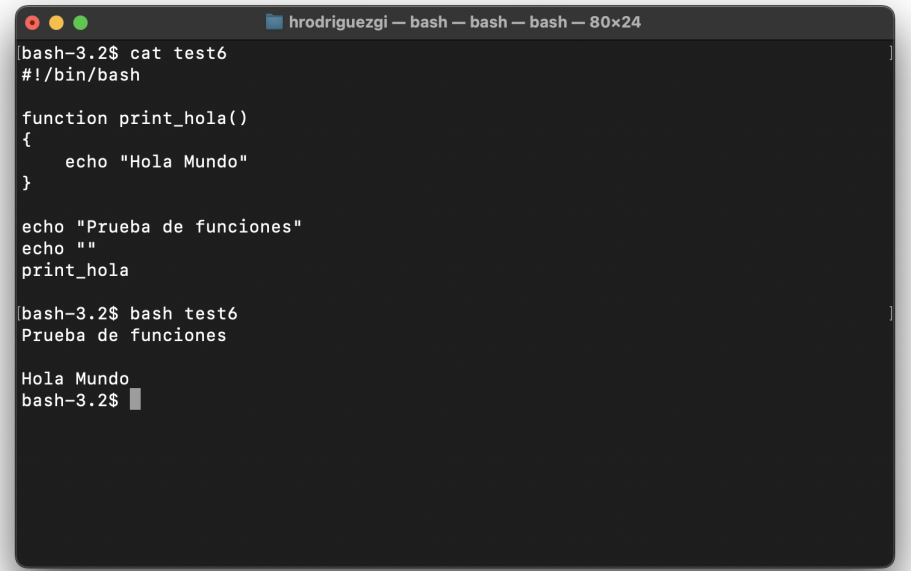
```
function_name()
{
 comandos
}
```

O también se puede escribir de la siguiente forma:

```
function function_name
{
 comandos
}
```

Funciones

Se debe tener presente que definir una función no implica que se utilice en el script, por lo tanto en el momento de querer hacer uso de ella, deberemos invocarla:



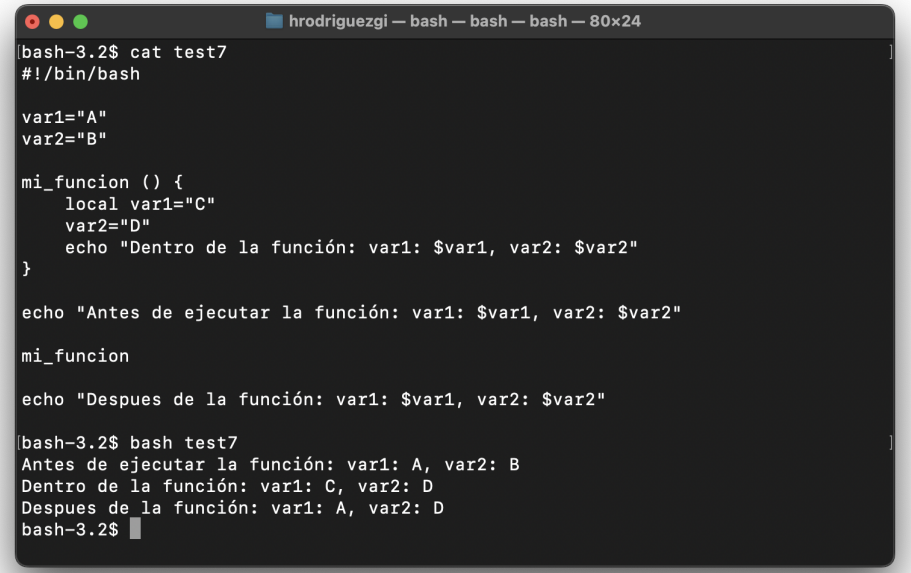
Funciones - Variables

Las variables en los scripts son por defecto globales en el script, sin embargo podemos definirlas de forma local si lo requerimos:

```
function_name()
{
 local var1='C'
}
```

De esta forma, la variable `var1` solo tendrá alcance dentro de la función.

Funciones - Variables



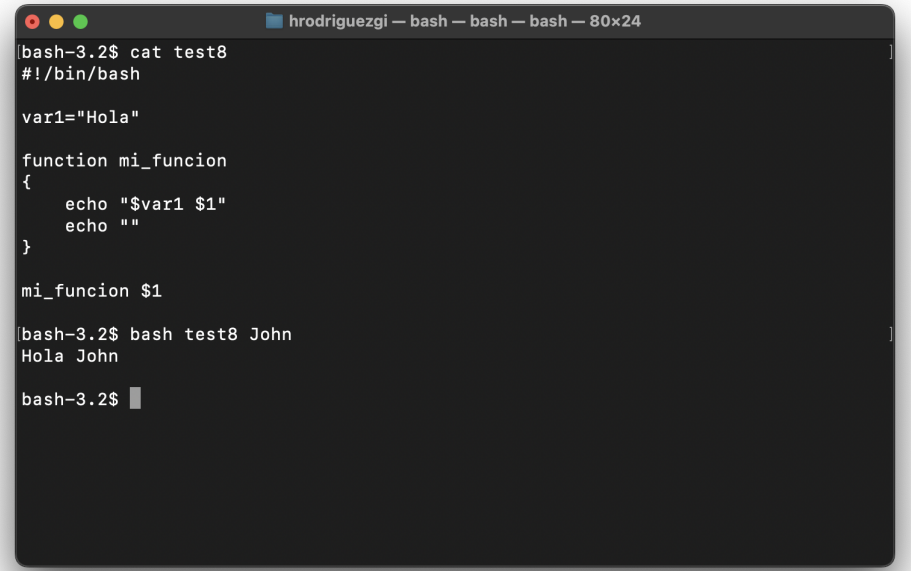
Funciones - Argumentos

Las funciones son capaces también de tomar los valores de los argumentos al momento de ejecutarse un script, bajo la misma lógica: $1, $2, ... , $n

```
function_name()
{
 var1=$1
 local var2=$2
}
```

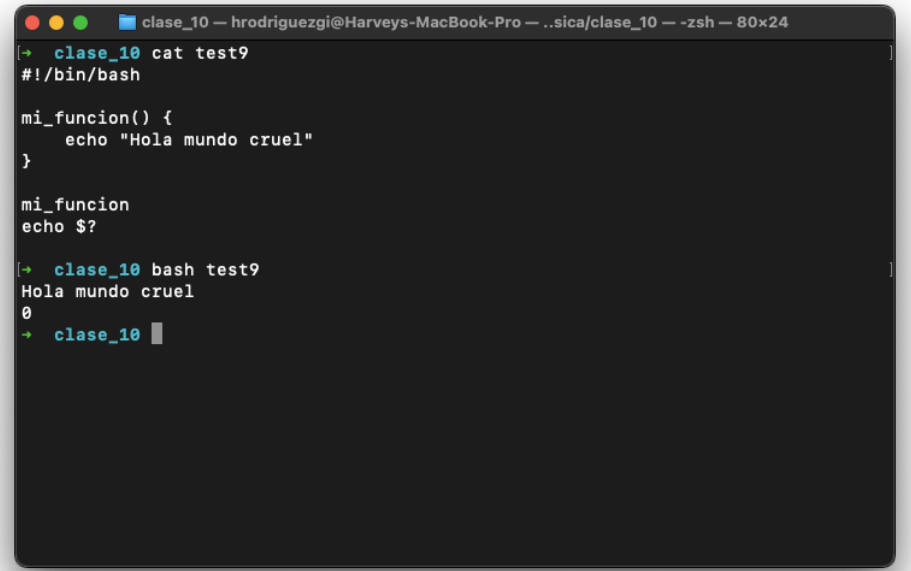
Funciones - Argumentos

Al igual que lo visto anteriormente, podemos utilizar la cantidad de argumentos que requiera nuestro script y ser llamados desde las funciones, incluso podemos hacer uso de $@ que utilizaría todos los argumentos recibidos:



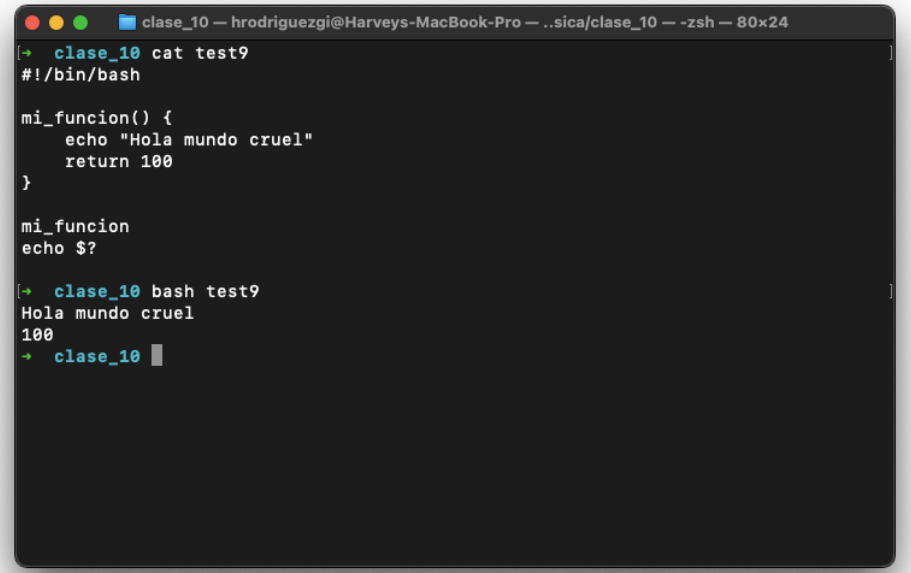
Funciones - Salida

Las funciones a diferencia de los lenguajes no permiten devolver un valor cuando son invocadas. Al igual que lo visto anteriormente retornan el estado del último comando ejecutado en ella



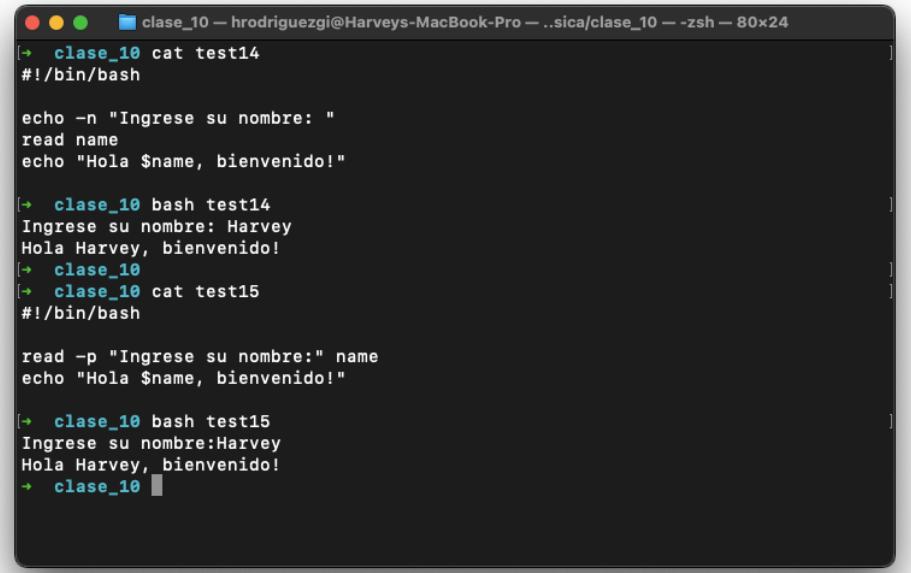
Funciones - Salida

Sin embargo, al igual que en los bash script, podemos personalizar el código de salida con la palabra `return`



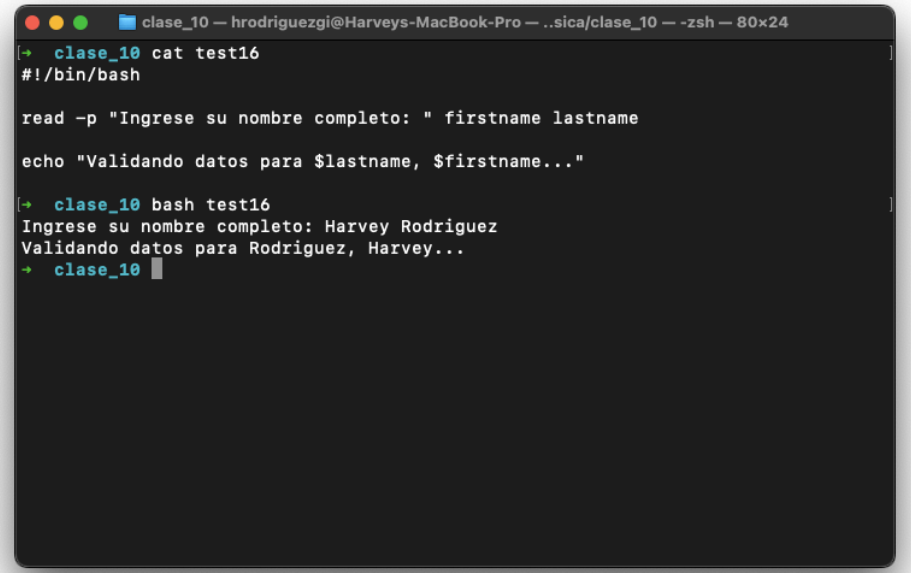
Interactuando con el usuario

El comando `read` nos permite interactuar con el usuario en un script, esperando que ingrese un dato a través del teclado:



Interactuando con el usuario

También se pueden leer múltiples argumentos al momento de ejecutar el programa:



Interactuando con el usuario

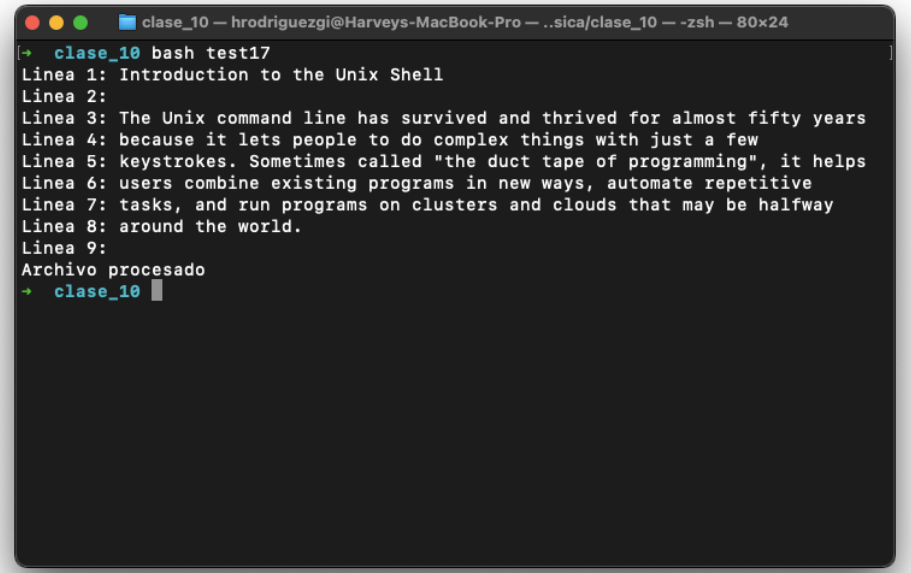
También se pueden leer desde un archivo por medio del comando `read` en conjunto con `while`:

clase_10 — hrodriguezgi@Harveys-MacBook-Pro — ..sica/clase_10 — -zsh — 80×24

```
→  clase_10 cat course.txt
Introduction to the Unix Shell

The Unix command line has survived and thrived for almost fifty years
because it lets people to do complex things with just a few
keystrokes. Sometimes called "the duct tape of programming", it helps
users combine existing programs in new ways, automate repetitive
tasks, and run programs on clusters and clouds that may be halfway
around the world.

→  clase_10 cat test17
#!/bin/bash

count=1
cat course.txt | while read line
do
    echo "Linea $count: $line"
    count=$[ $count + 1 ]
done
echo "Archivo procesado"

→  clase_10
```

Interactuando con el usuario

Esta es la salida del programa anterior



Vamos a practicar

- Crear un script que calcule el factorial de un número que ingrese el usuario
- Crear un script que reciba pregunte:
  - Número 1
  - Número 2
  - Operación a realizar

  La salida deberá ser el resultado de la operación. Utilizar funciones para cada operación.