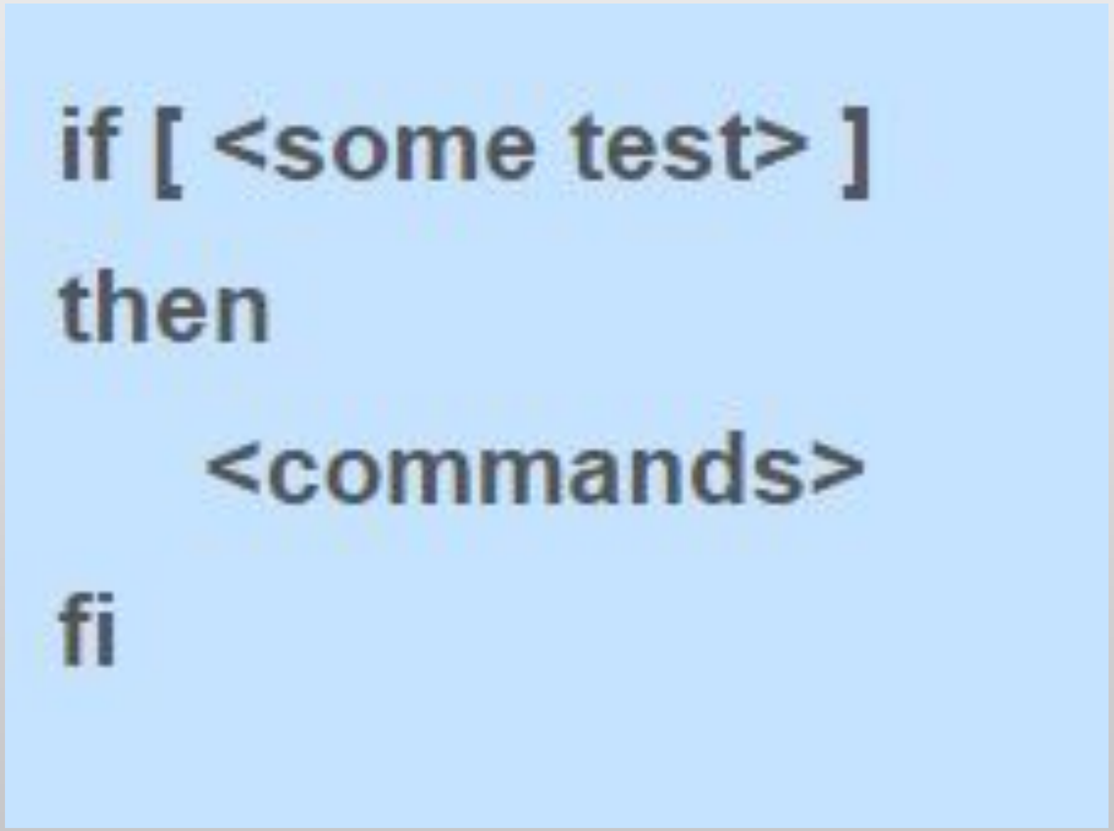
OS Lab 4- Shell Scripting

Instructor

Numan Shafi

Numan.shafi@uet.edu.pk

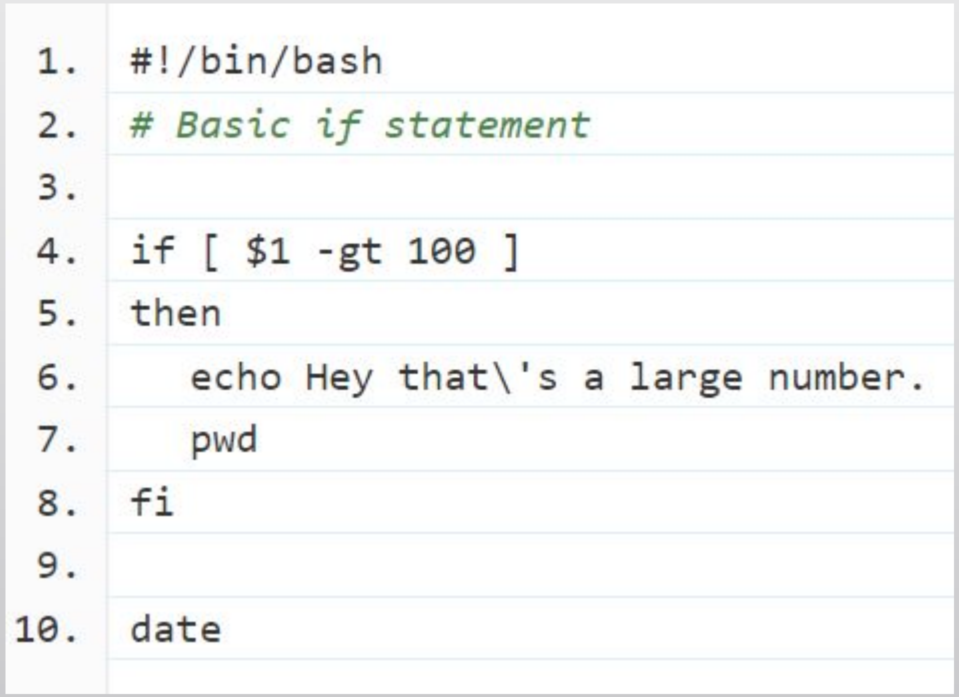
Decisions



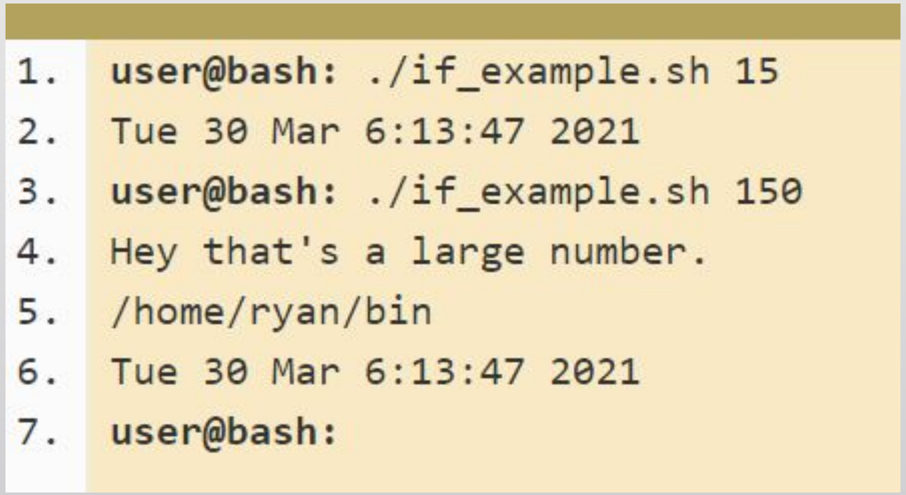
```
if [ <some test> ]
then
    <commands>
fi
```

```
If ()
{//start
//////
}//end
```

Example 1

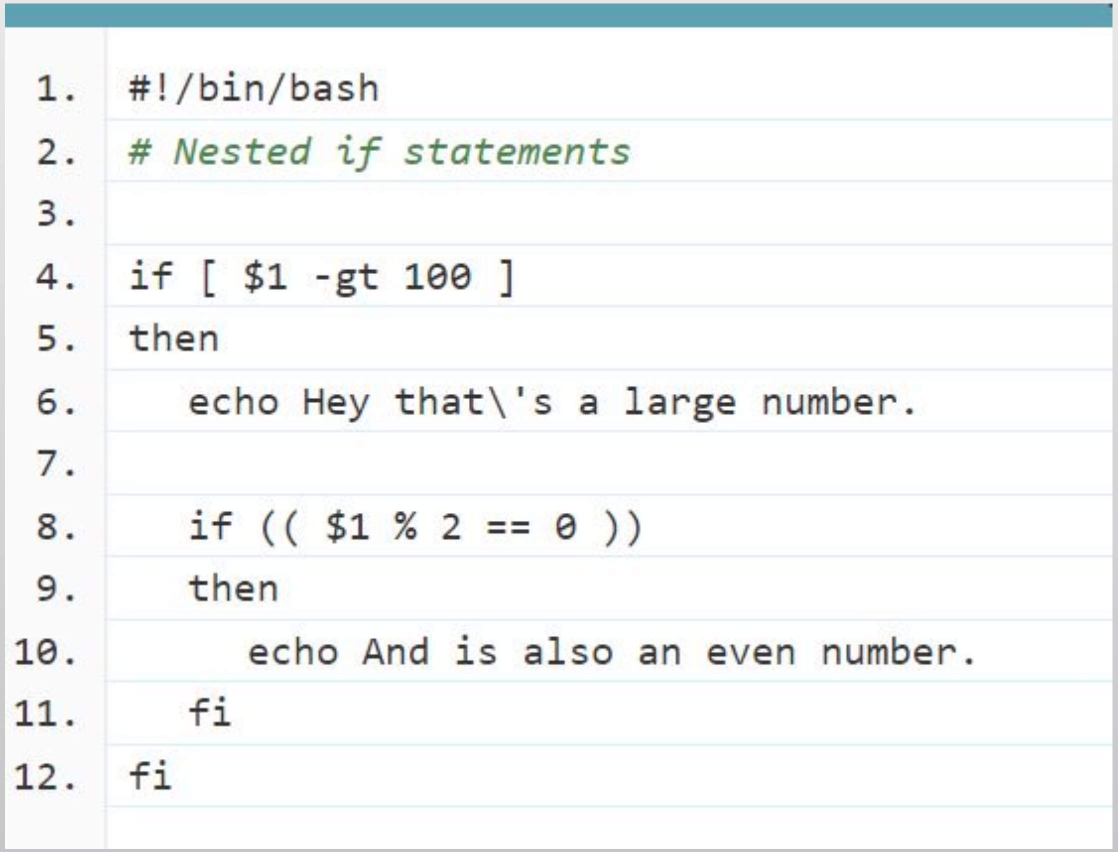


```bash
#!/bin/bash
# Basic if statement

if [ $1 -gt 100 ]
then
   echo Hey that\'s a large number.
   pwd
fi

date
```



```
user@bash: ./if_example.sh 15
Tue 30 Mar 6:13:47 2021
user@bash: ./if_example.sh 150
Hey that's a large number.
/home/ryan/bin
Tue 30 Mar 6:13:47 2021
user@bash:
```

Nested If



```
#!/bin/bash
# Nested if statements

if [ $1 -gt 100 ]
then
   echo Hey that\'s a large number.

   if (( $1 % 2 == 0 ))
   then
      echo And is also an even number.
   fi
fi
```

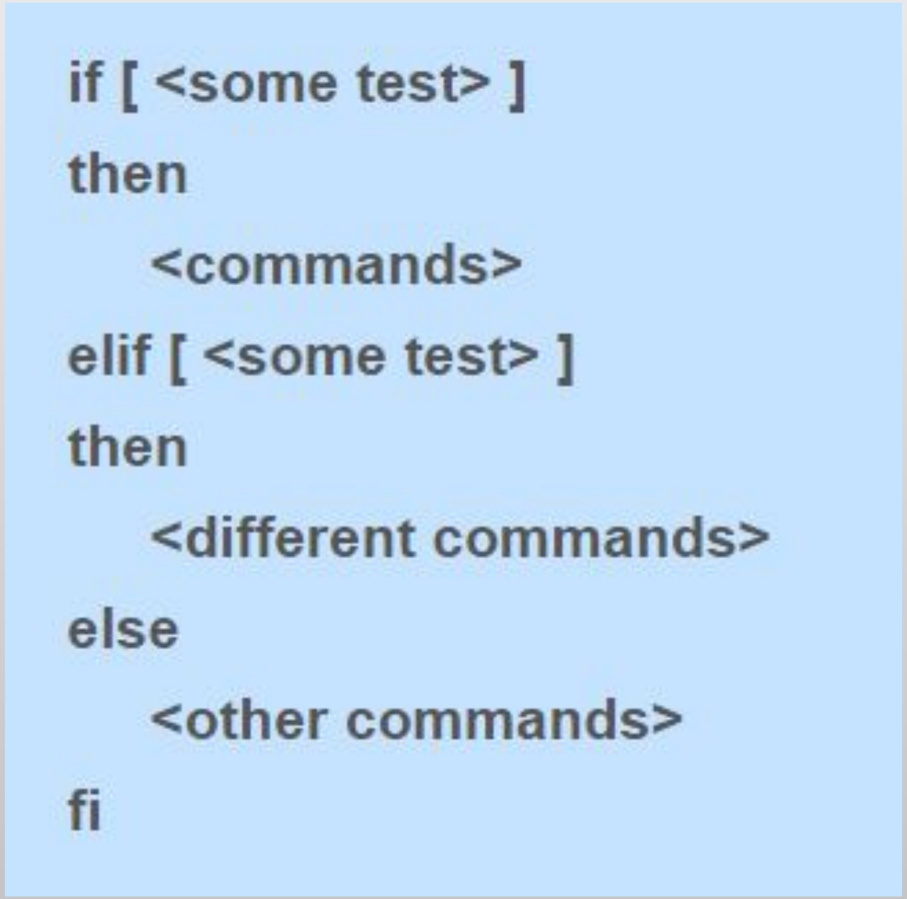
If-Else

```
if [ <some test> ]
then
    <commands>
else
    <other commands>
fi
```

Example

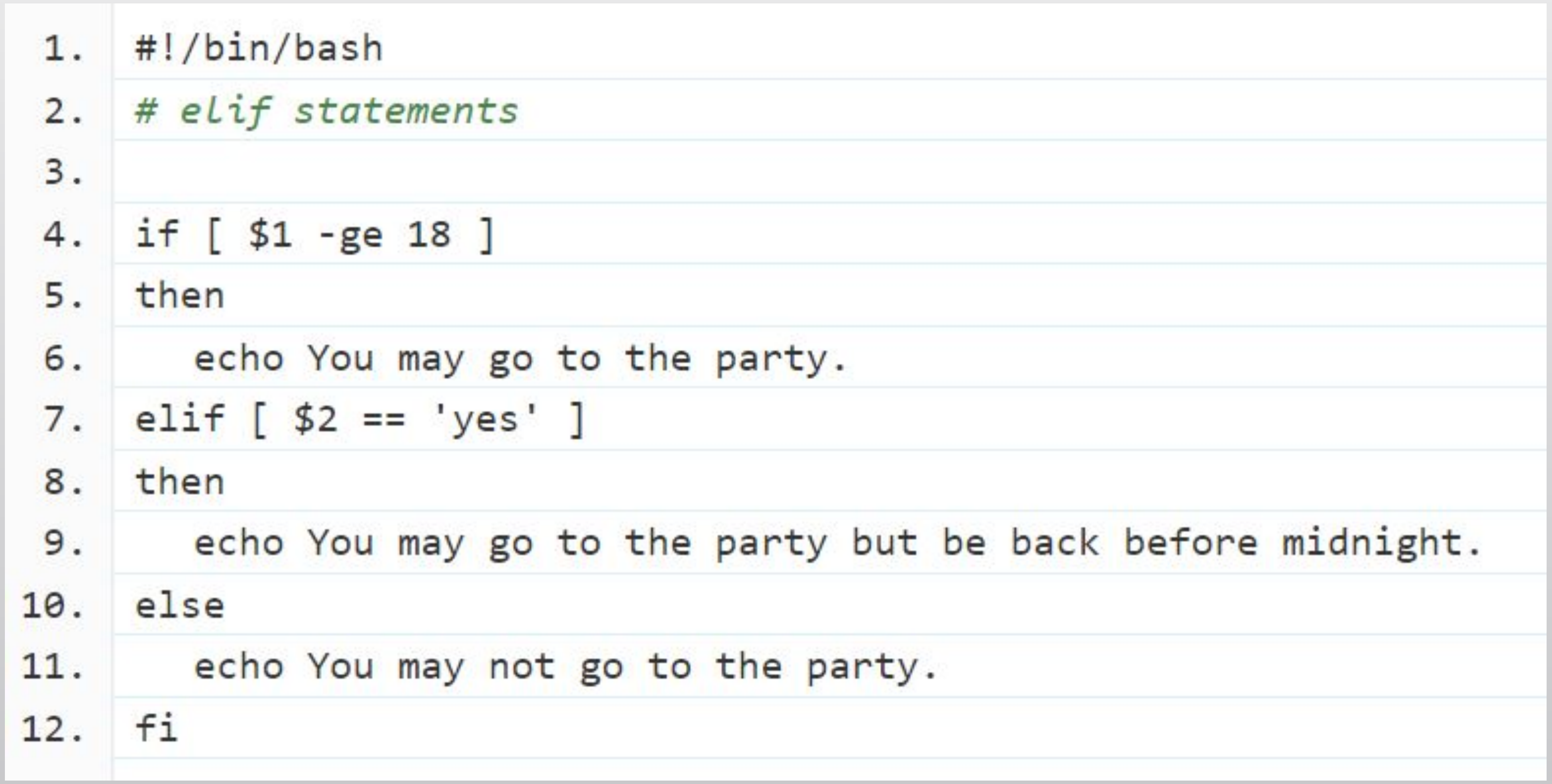
```bash
#!/bin/bash
# else example
if [ $# -eq 1 ]
then
    echo happy
else
    echo sad
fi
```

If-elif-else



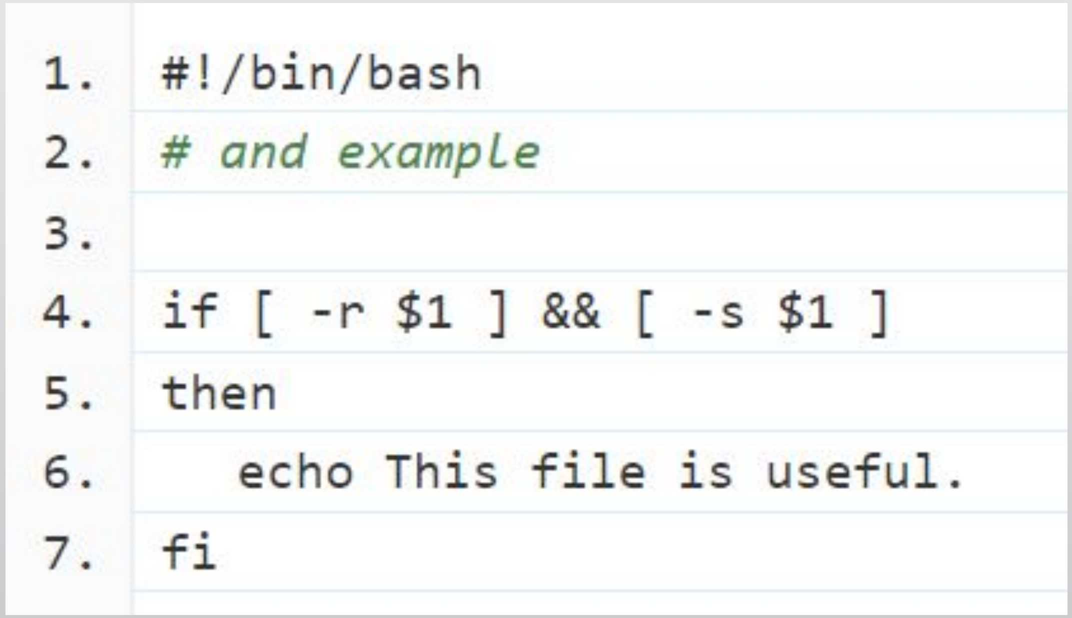
```
if [ <some test> ]
then
    <commands>
elif [ <some test> ]
then
    <different commands>
else
    <other commands>
fi
```

Example

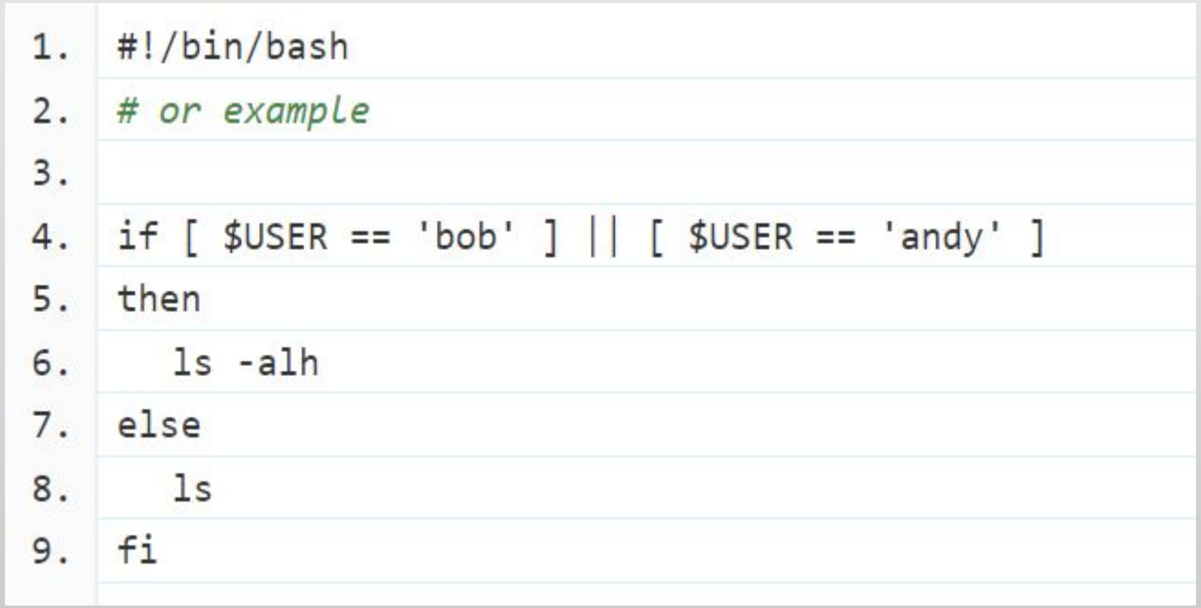


```
#!/bin/bash
# elif statements

if [ $1 -ge 18 ]
then
   echo You may go to the party.
elif [ $2 == 'yes' ]
then
   echo You may go to the party but be back before midnight.
else
   echo You may not go to the party.
fi
```

Boolean operations

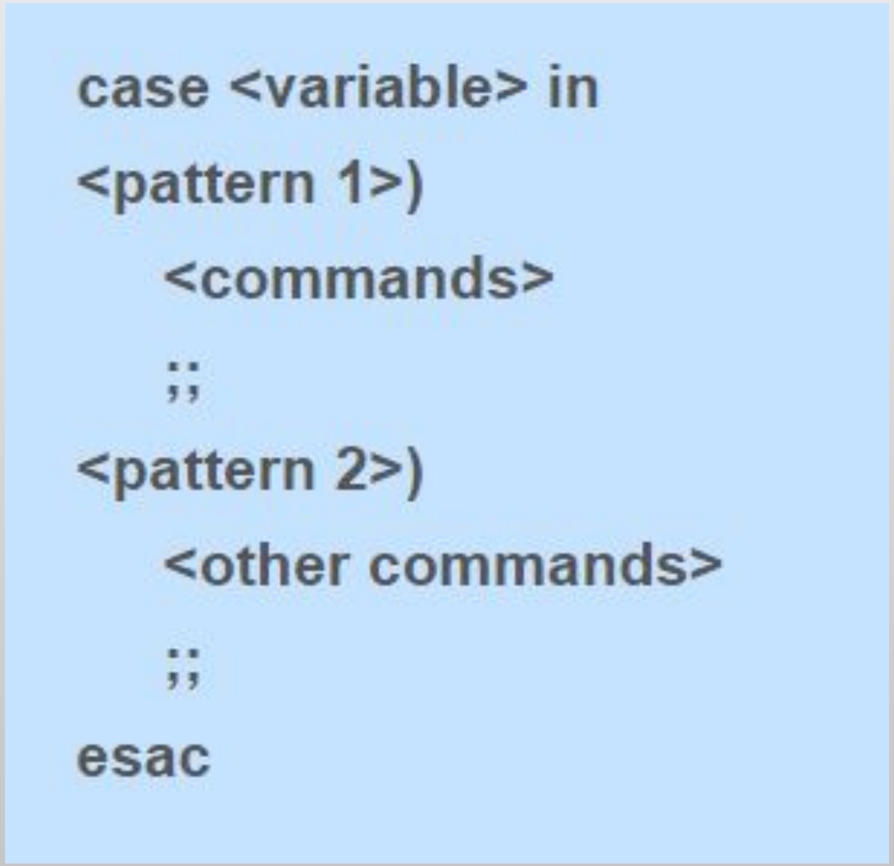


```bash
#!/bin/bash
# and example

if [ -r $1 ] && [ -s $1 ]
then
   echo This file is useful.
fi
```



```bash
#!/bin/bash
# or example

if [ $USER == 'bob' ] || [ $USER == 'andy' ]
then
   ls -alh
else
   ls
fi
```

Case Statement

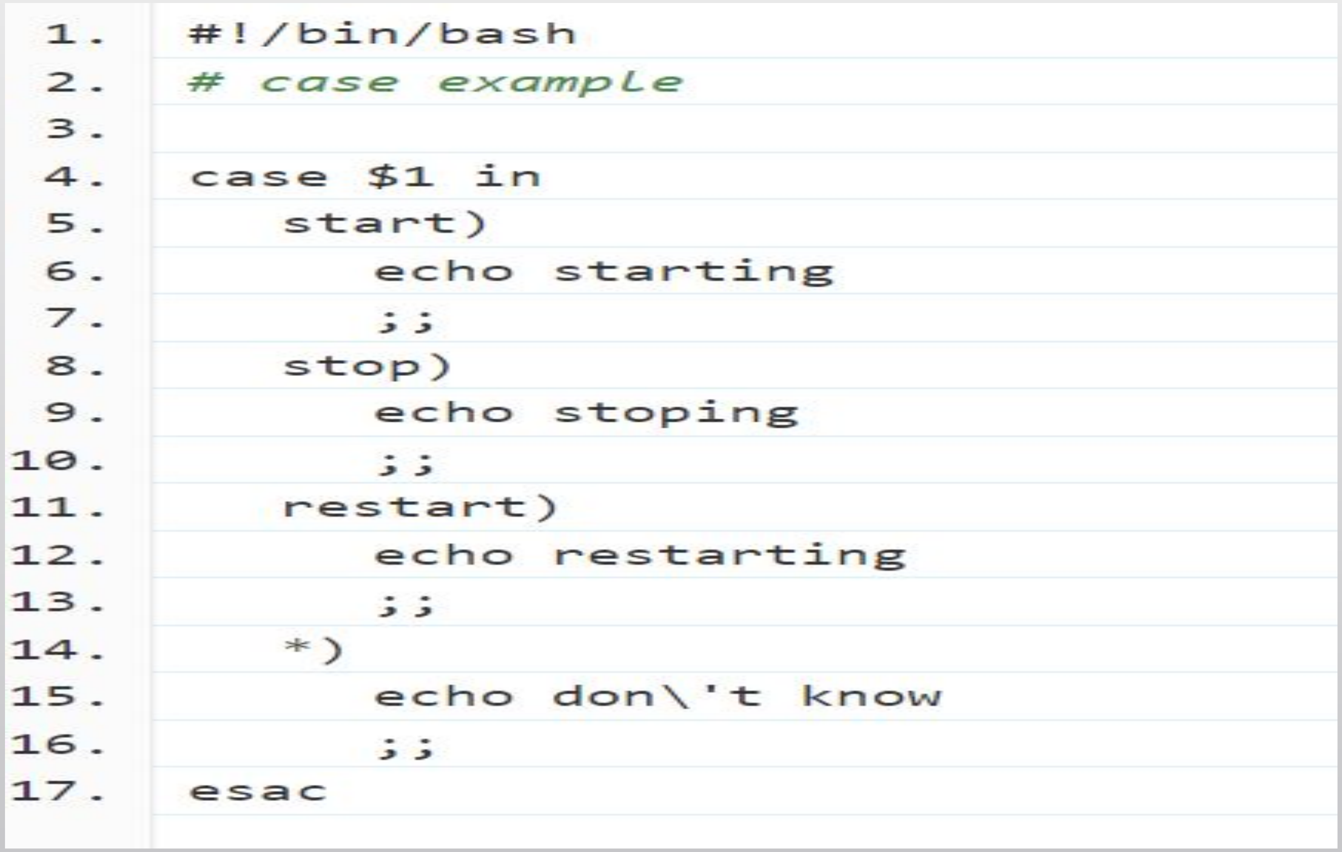


```
case <variable> in
<pattern 1>)
    <commands>
    ;;
<pattern 2>)
    <other commands>
    ;;
esac
```

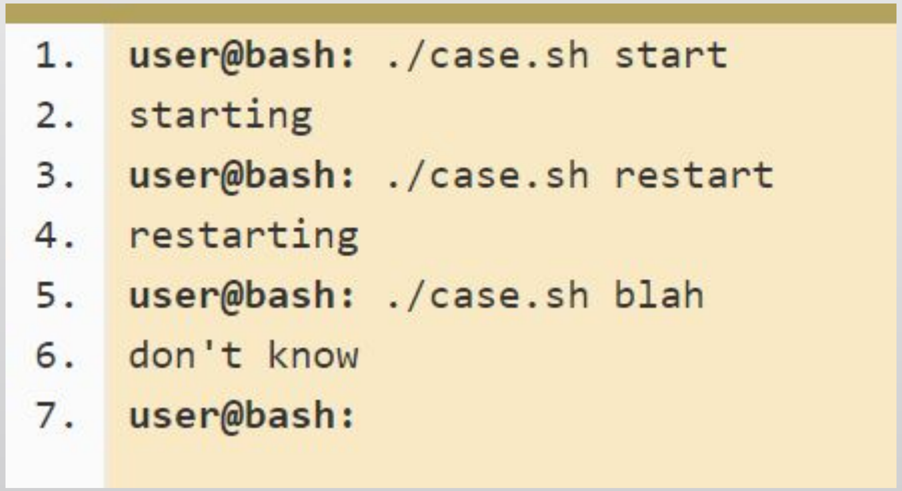
Case Statement

```
#!/bin/bash
DEPARTMENT="Computer Science"

echo -n "Your DEPARTMENT is "

case $DEPARTMENT in
  "Computer Science")
    echo -n "Computer Science"
    ;;
   "Electrical and Electronics Engineering" | "Electrical Engineering")
    echo -n "Electrical and Electronics Engineering or Electrical Engineering"
    ;;
  "Information Technology" | "Electronics and Communication")
    echo -n "Information Technology or Electronics and Communication"
    ;;
  *)
    echo -n "Invalid"
    ;;
esac
```

Example



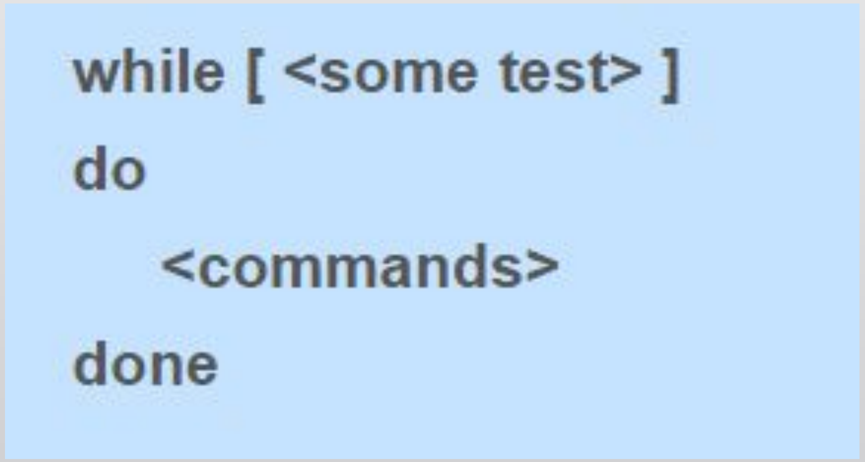
```bash
#!/bin/bash
# case example

case $1 in
    start)
        echo starting
        ;;
    stop)
        echo stoping
        ;;
    restart)
        echo restarting
        ;;
    *)
        echo don\'t know
        ;;
esac
```



```
user@bash: ./case.sh start
starting
user@bash: ./case.sh restart
restarting
user@bash: ./case.sh blah
don't know
user@bash:
```

LOOPS

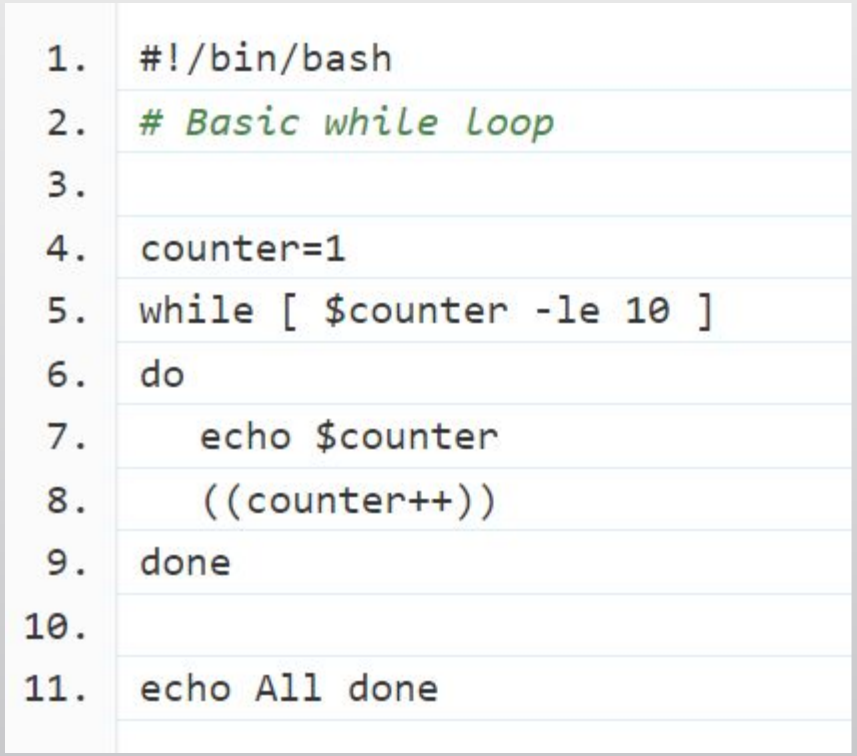
While Loop



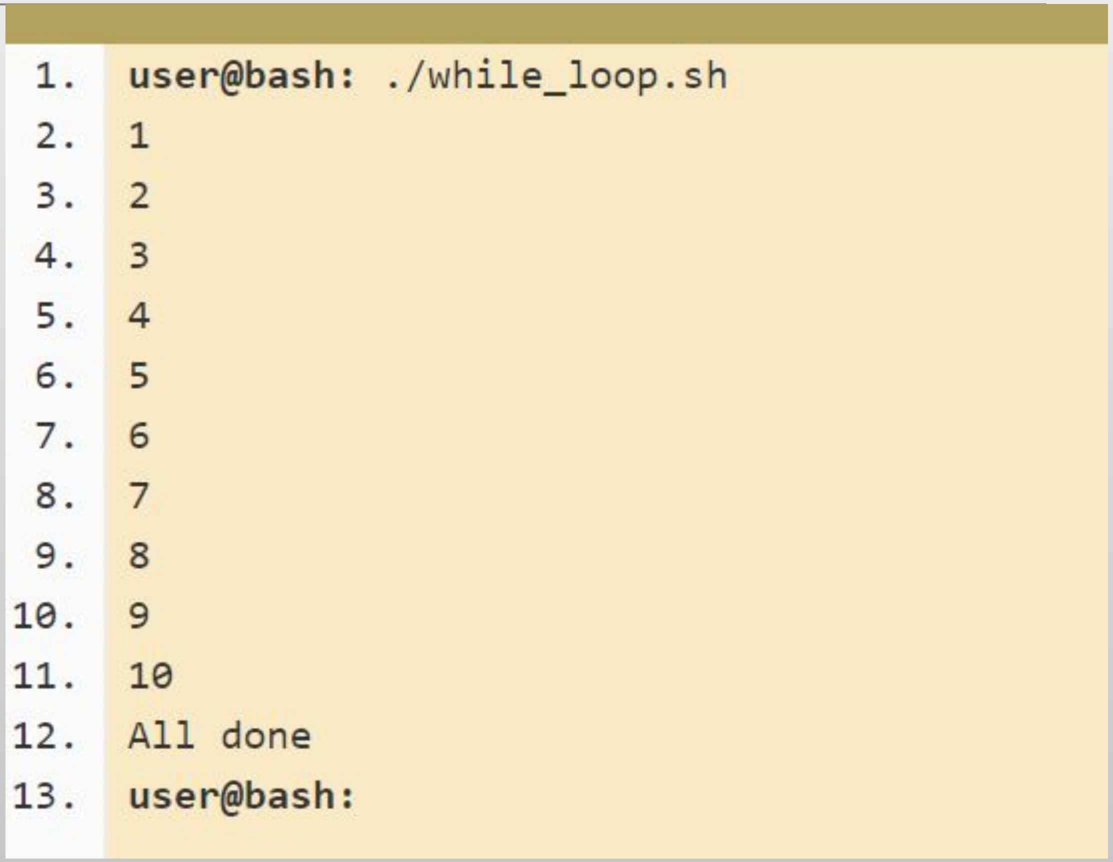
```
while [ <some test> ]
do
    <commands>
done
```

```
While (stmt)
{//start

//////

}//end
```

Example

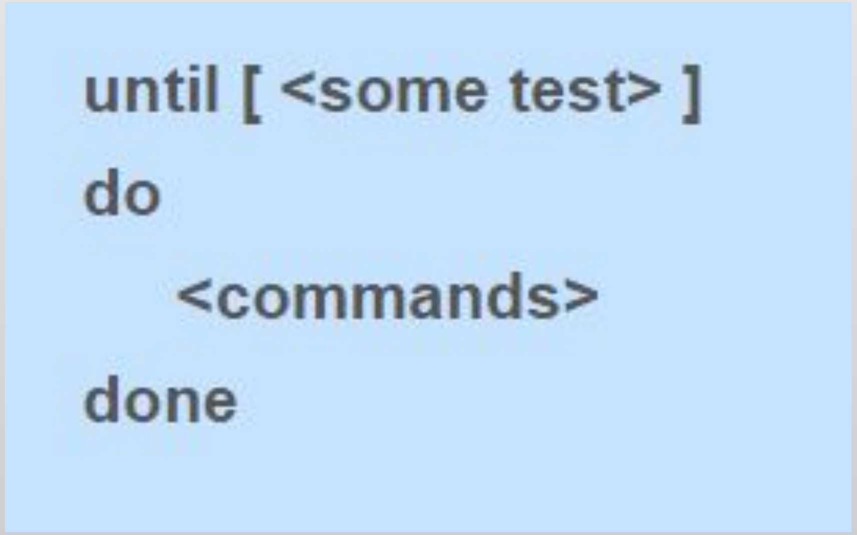


```
#!/bin/bash
# Basic while loop

counter=1
while [ $counter -le 10 ]
do
   echo $counter
   ((counter++))
done

echo All done
```



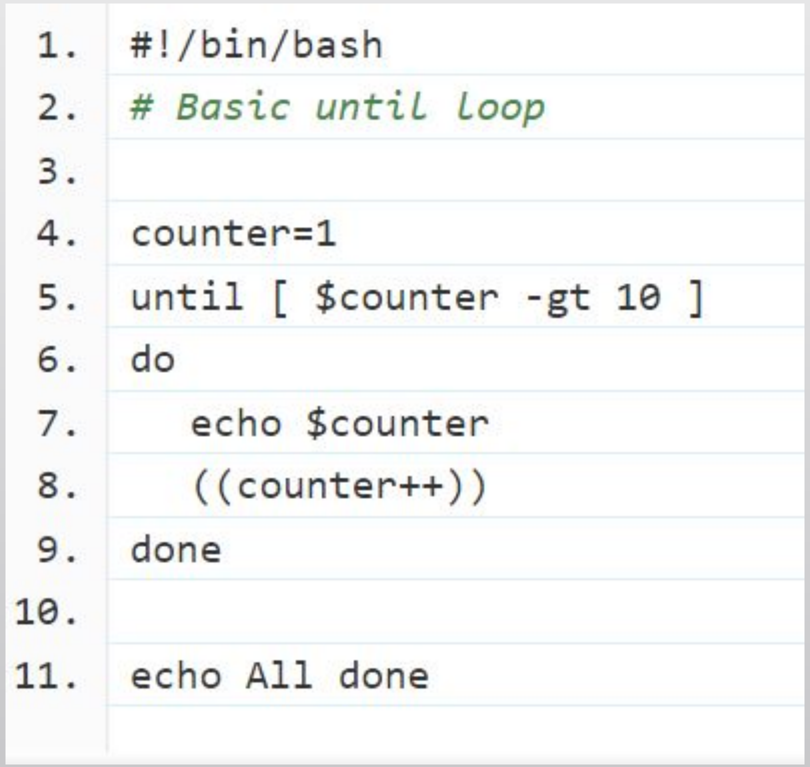
```
user@bash: ./while_loop.sh
1
2
3
4
5
6
7
8
9
10
All done
user@bash:
```

Until Loop



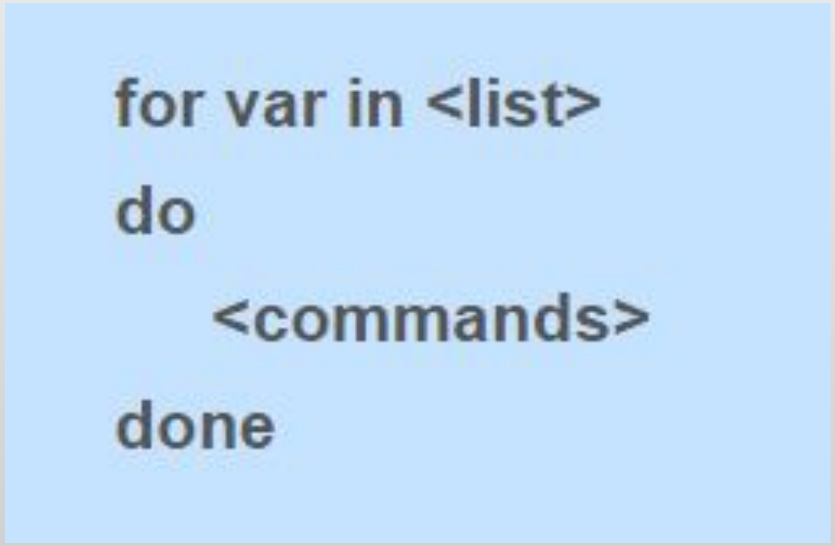
```
until [ <some test> ]
do
    <commands>
done
```

Example



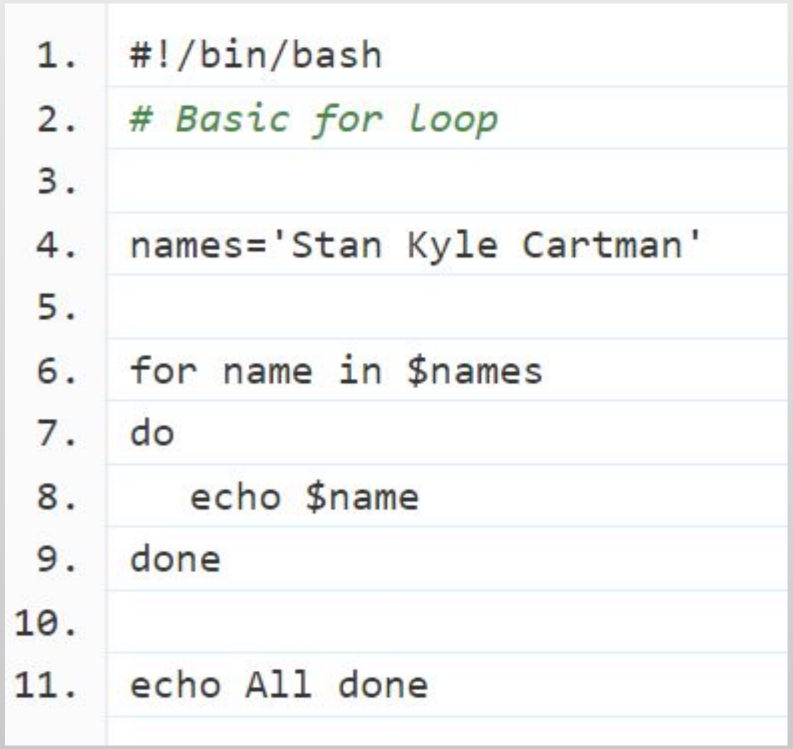
```bash
#!/bin/bash
# Basic until loop

counter=1
until [ $counter -gt 10 ]
do
   echo $counter
   ((counter++))
done

echo All done
```

For Loop

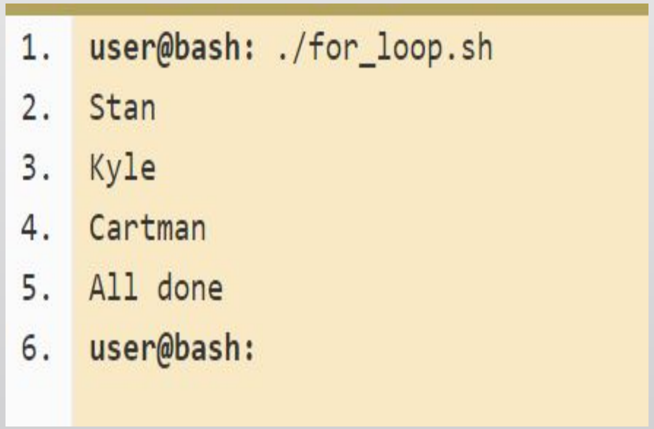


```
for var in <list>
do
    <commands>
done
```

```
for (int i=0; i<10;i++)
{//start
//////
}//end
```

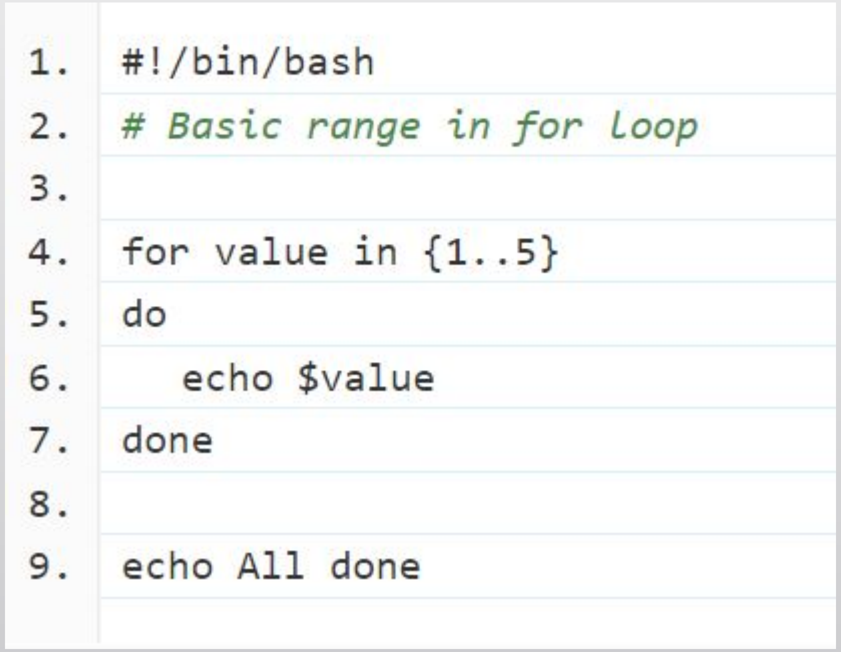


```
#!/bin/bash
# Basic for loop

names='Stan Kyle Cartman'

for name in $names
do
   echo $name
done

echo All done
```

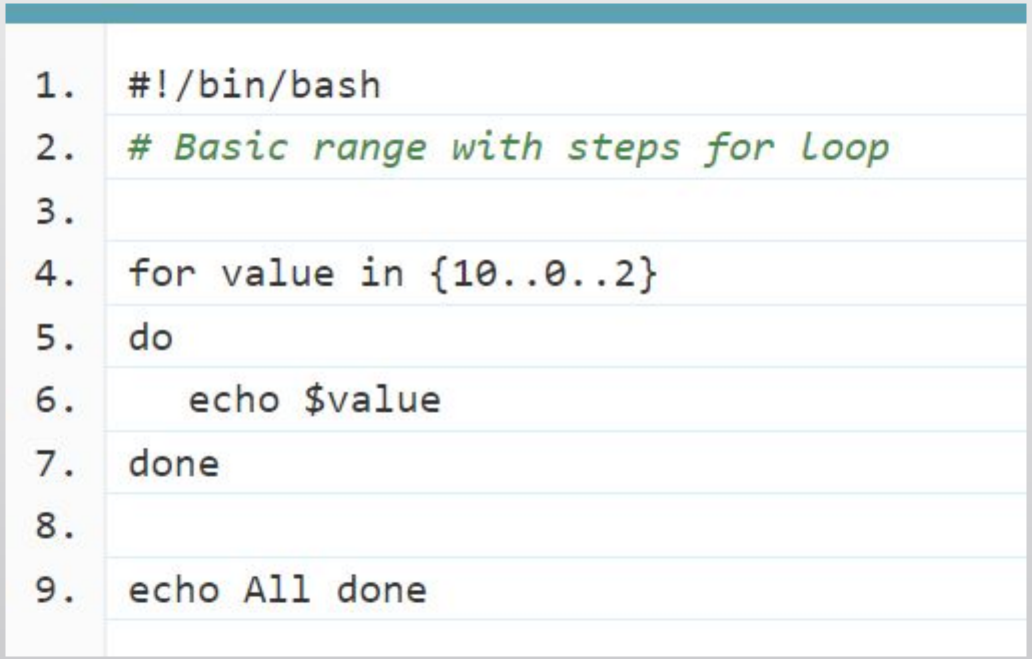


```
user@bash: ./for_loop.sh
Stan
Kyle
Cartman
All done
user@bash:
```

Ranges in for loop

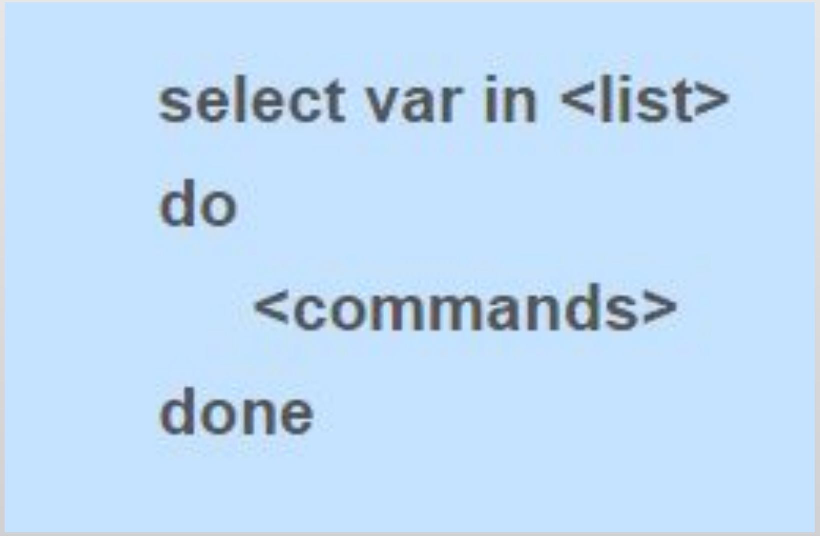


```bash
#!/bin/bash
# Basic range in for loop

for value in {1..5}
do
   echo $value
done

echo All done
```



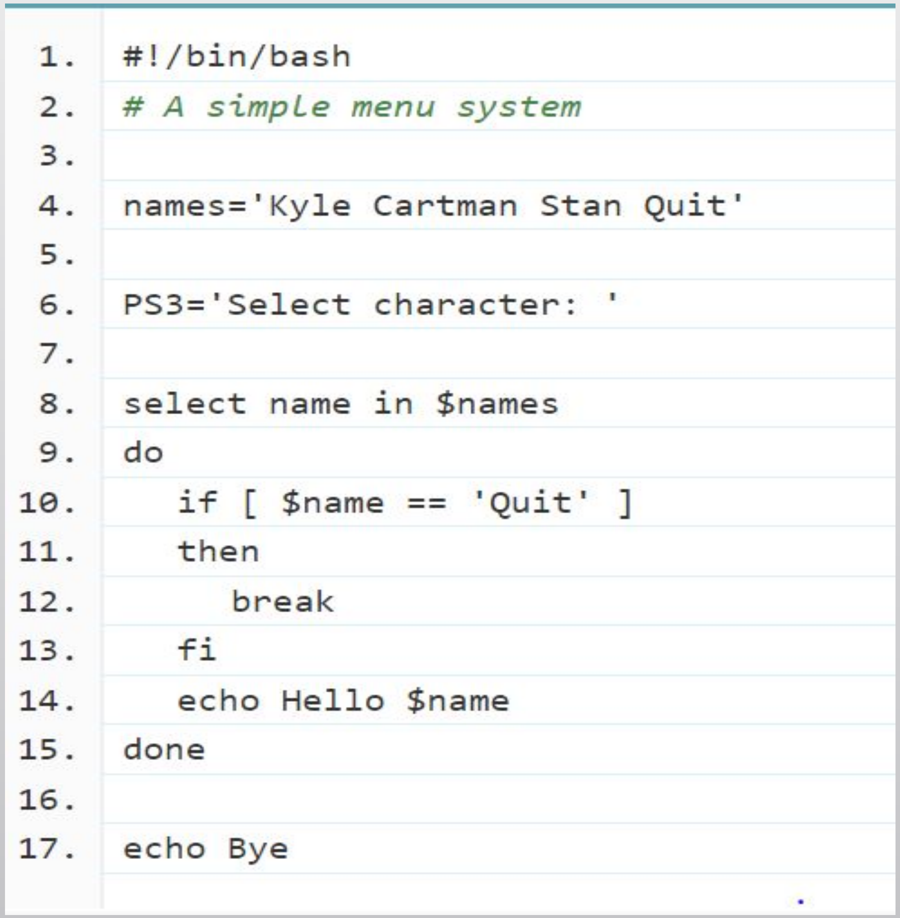
```bash
#!/bin/bash
# Basic range with steps for loop

for value in {10..0..2}
do
   echo $value
done

echo All done
```

Select Statement

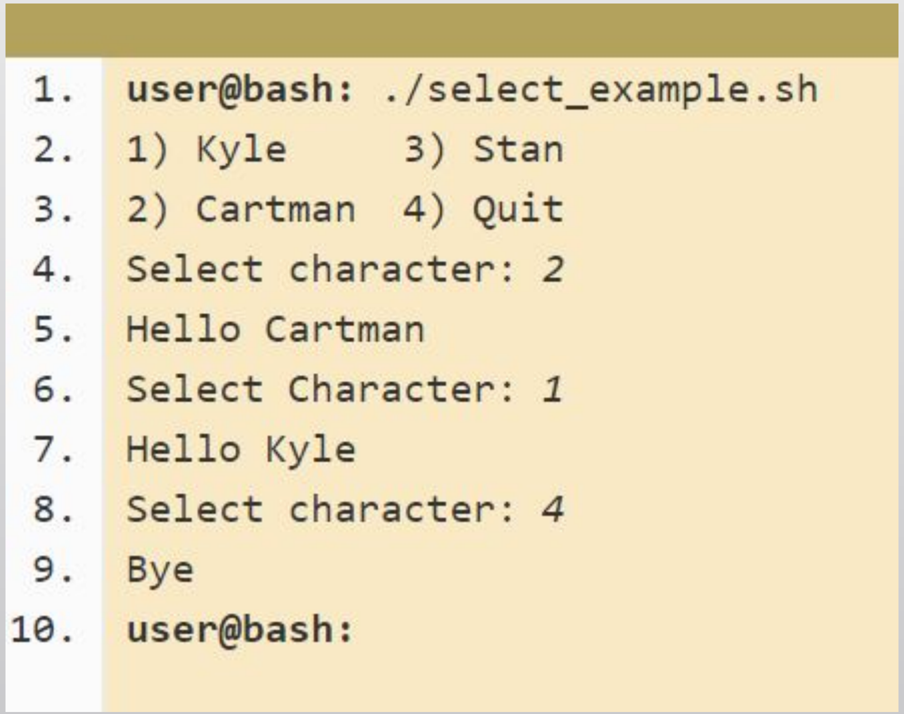


```
select var in <list>
do
    <commands>
done
```

Example



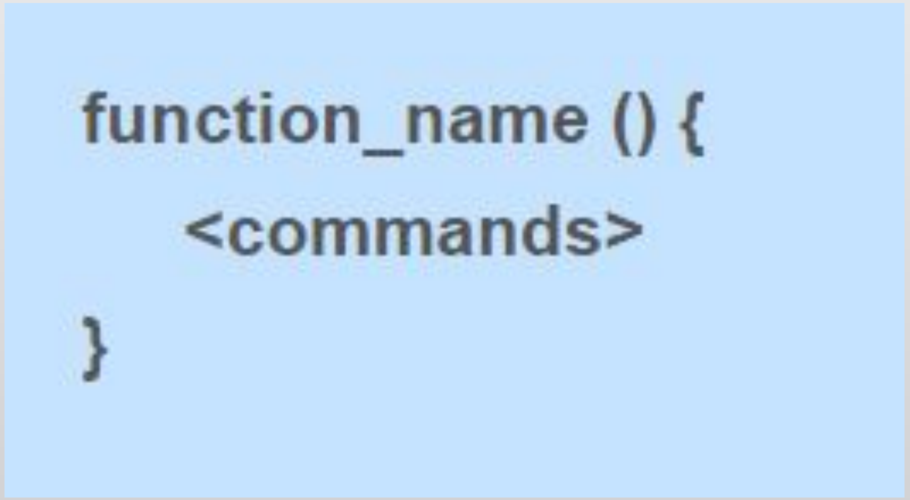
```bash
#!/bin/bash
# A simple menu system

names='Kyle Cartman Stan Quit'

PS3='Select character: '

select name in $names
do
   if [ $name == 'Quit' ]
   then
      break
   fi
   echo Hello $name
done

echo Bye
```



```
user@bash: ./select_example.sh
1) Kyle     3) Stan
2) Cartman  4) Quit
Select character: 2
Hello Cartman
Select Character: 1
Hello Kyle
Select character: 4
Bye
user@bash:
```

Functions

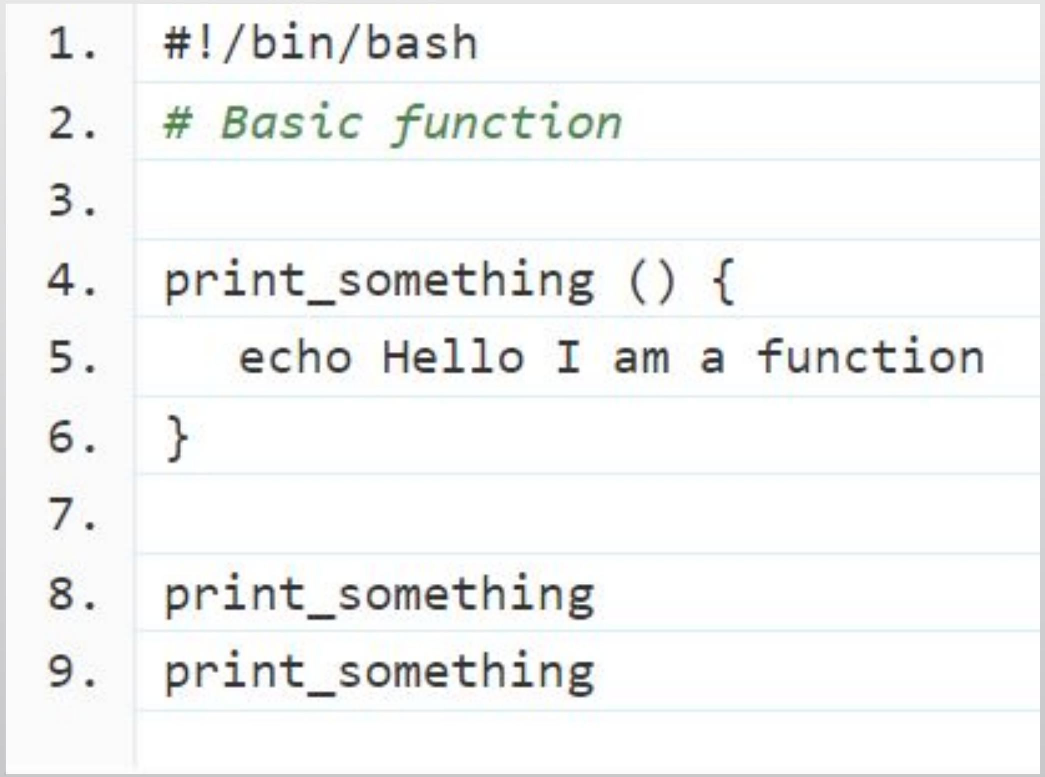
Simple function syntax



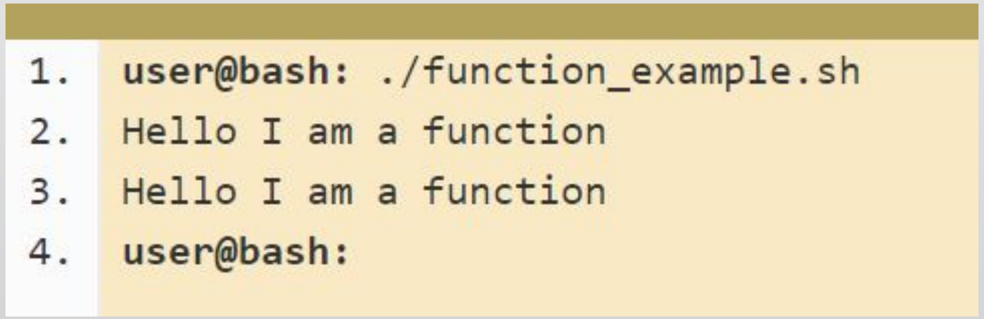
```
function_name () {
    <commands>
}
```

- In other programming languages it is common to have arguments passed to the function listed inside the brackets (). In Bash they are there only for decoration and you never put anything inside them.
- The function definition (the actual function itself) must appear in the script before any calls to the function.

Example

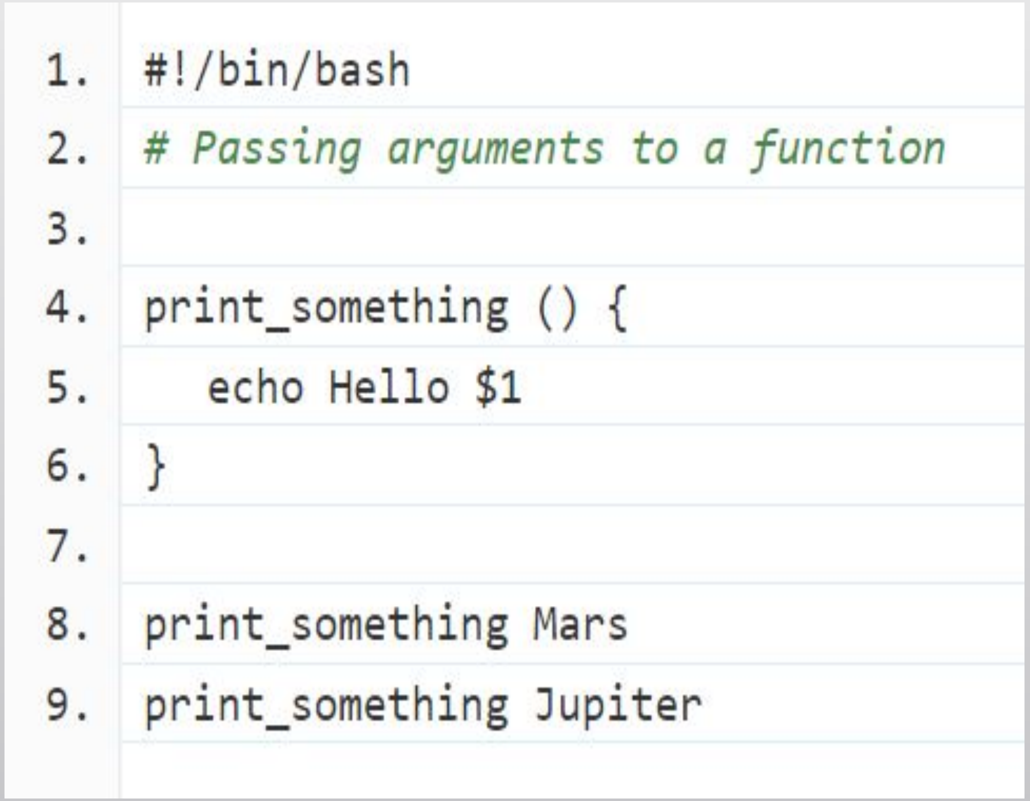


```bash
#!/bin/bash
# Basic function

print_something () {
   echo Hello I am a function
}

print_something
print_something
```

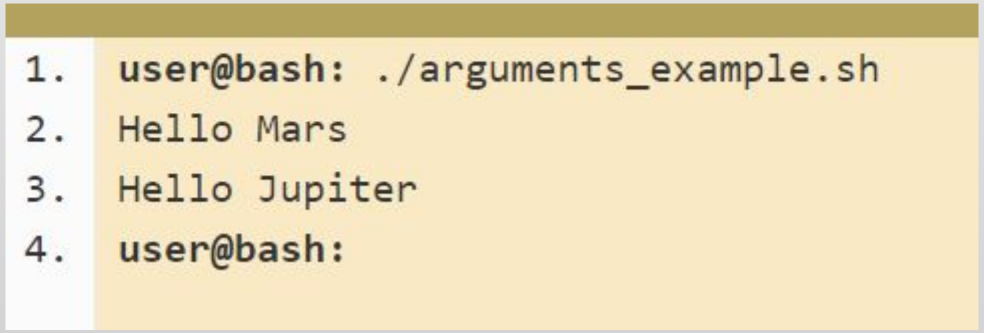


```
user@bash: ./function_example.sh
Hello I am a function
Hello I am a function
user@bash:
```

Passing Arguments

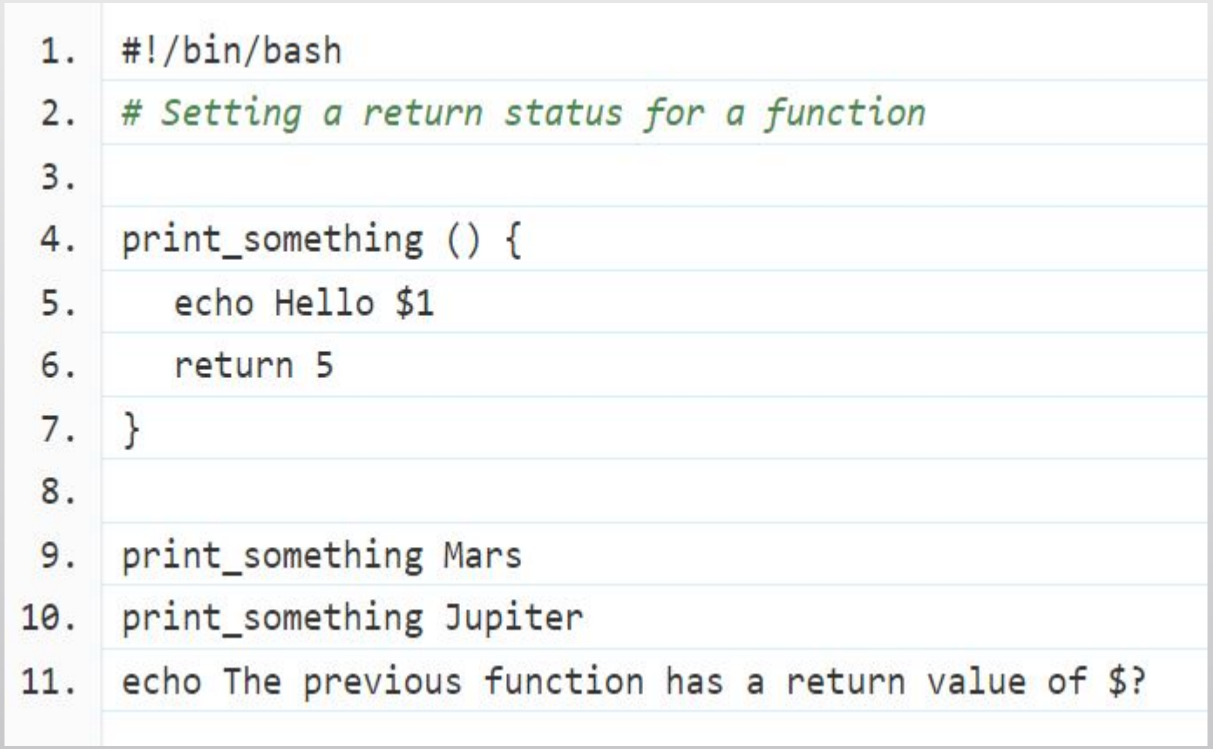


```
#!/bin/bash
# Passing arguments to a function

print_something () {
   echo Hello $1
}

print_something Mars
print_something Jupiter
```

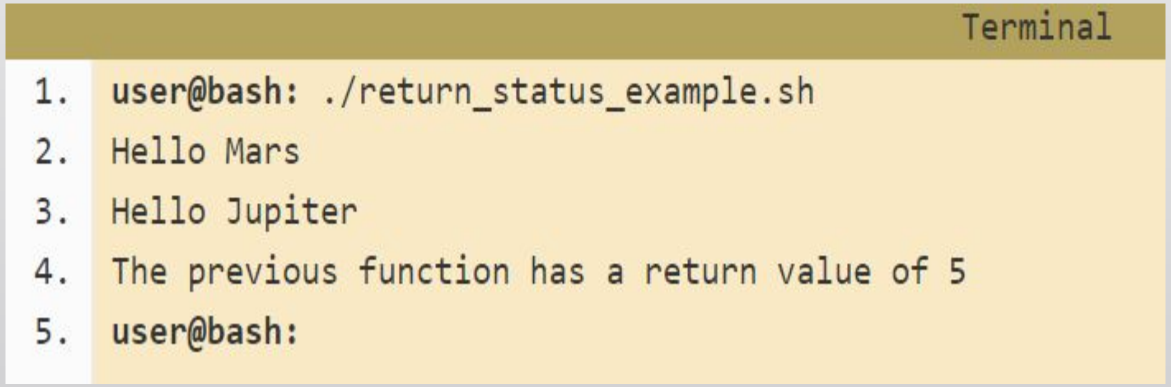


```
user@bash: ./arguments_example.sh
Hello Mars
Hello Jupiter
user@bash:
```

Returning Value

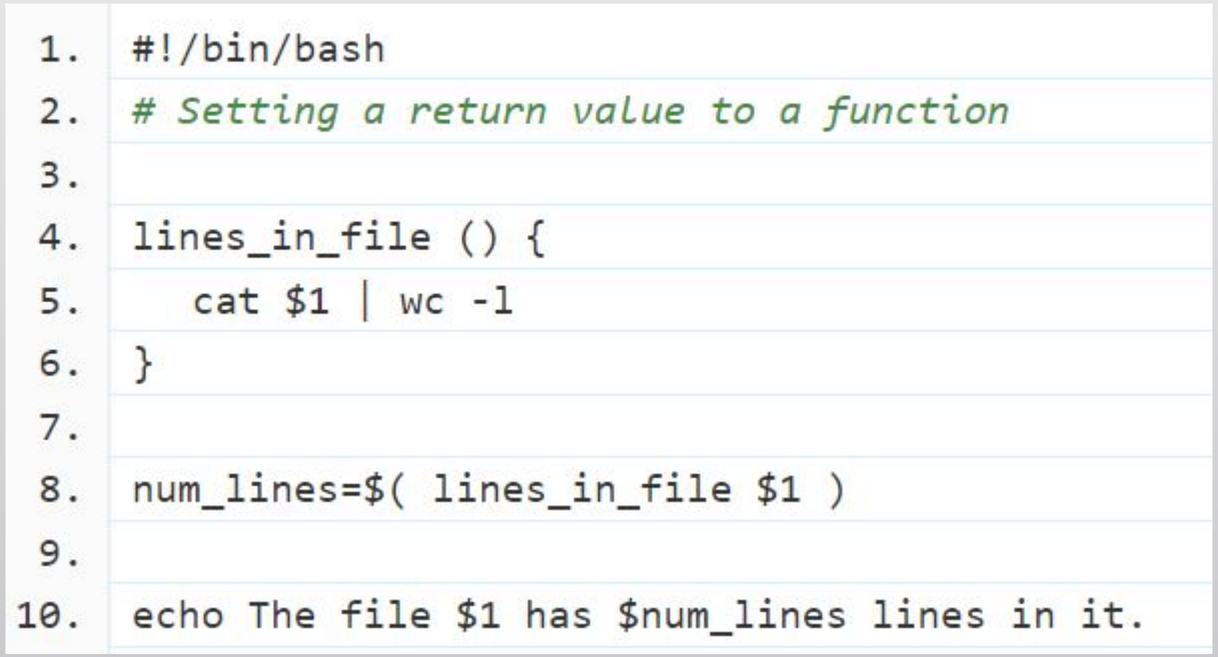


```bash
#!/bin/bash
# Setting a return status for a function

print_something () {
   echo Hello $1
   return 5
}

print_something Mars
print_something Jupiter
echo The previous function has a return value of $?
```

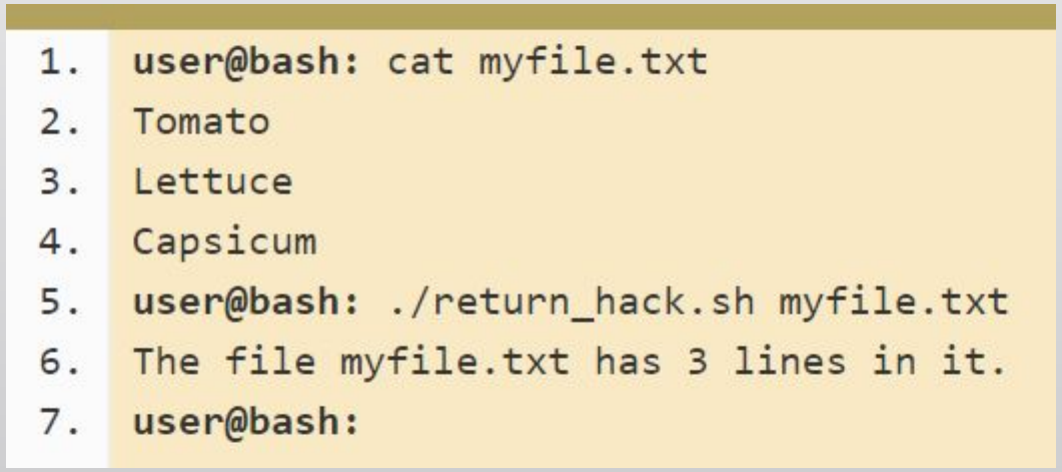


```
user@bash: ./return_status_example.sh
Hello Mars
Hello Jupiter
The previous function has a return value of 5
user@bash:
```

Returning Value (Alternate way)

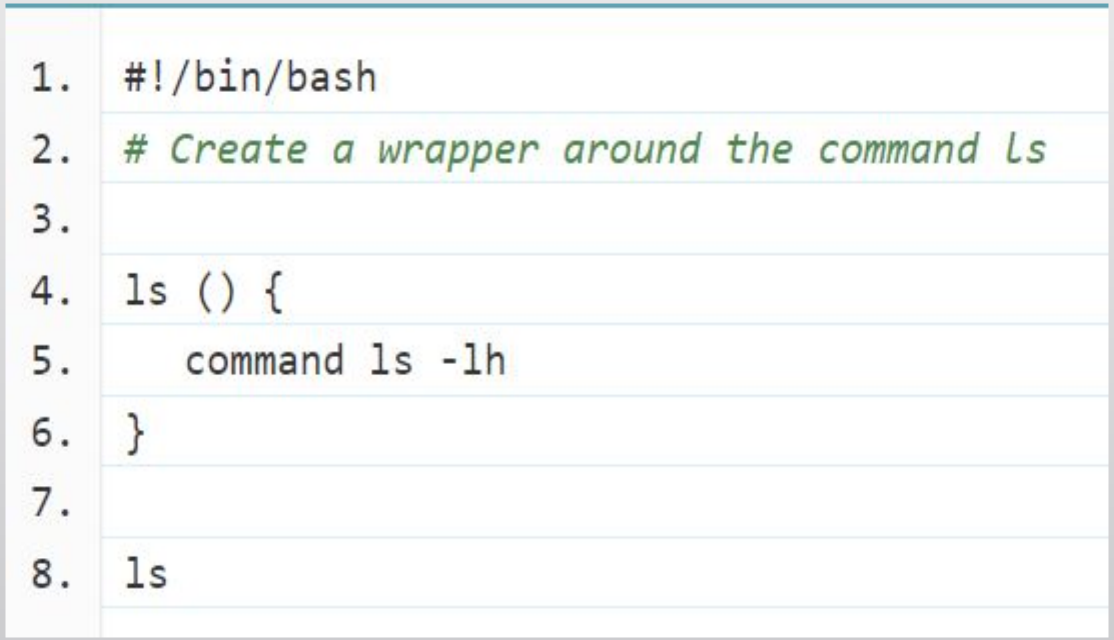


```bash
#!/bin/bash
# Setting a return value to a function

lines_in_file () {
    cat $1 | wc -l
}

num_lines=$( lines_in_file $1 )

echo The file $1 has $num_lines lines in it.
```



```
user@bash: cat myfile.txt
Tomato
Lettuce
Capsicum
user@bash: ./return_hack.sh myfile.txt
The file myfile.txt has 3 lines in it.
user@bash:
```

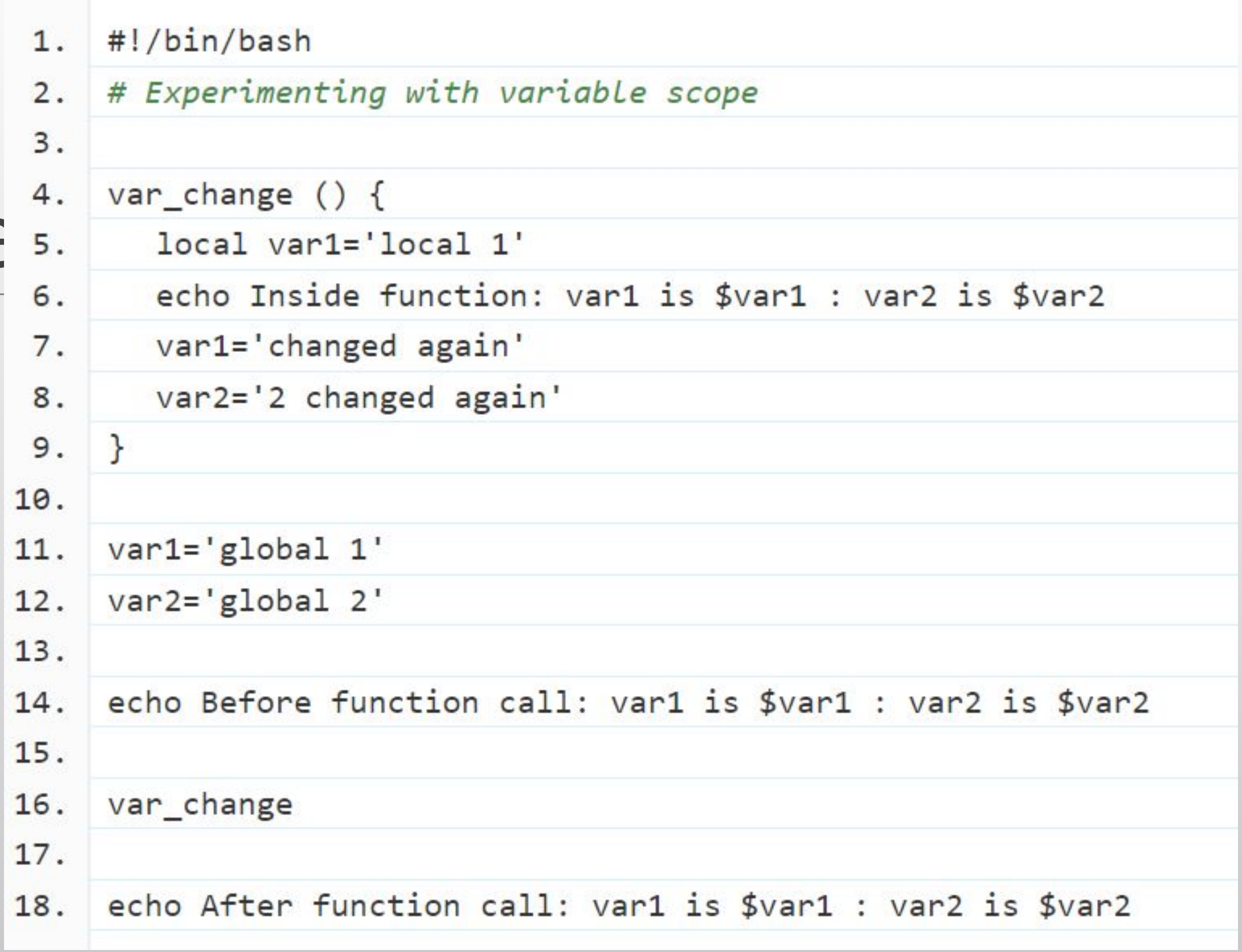
Overriding commands



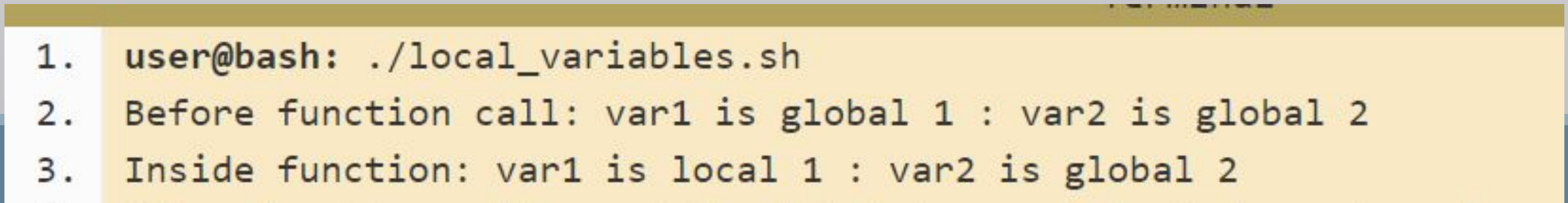
```bash
#!/bin/bash
# Create a wrapper around the command ls

ls () {
   command ls -lh
}

ls
```

Variable scope

local var_name=<var_value>

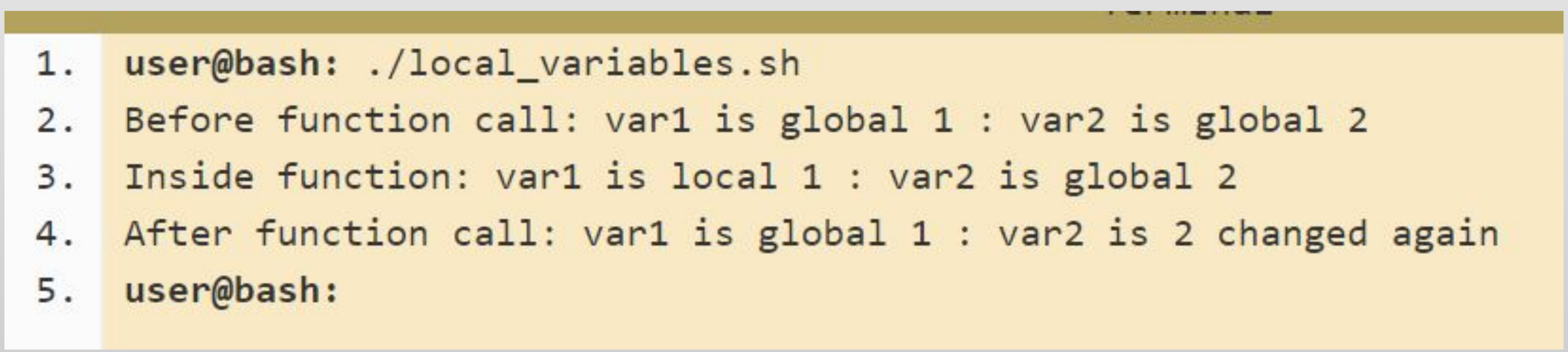


```bash
#!/bin/bash
# Experimenting with variable scope

var_change () {
  local var1='local 1'
  echo Inside function: var1 is $var1 : var2 is $var2
  var1='changed again'
  var2='2 changed again'
}

var1='global 1'
var2='global 2'

echo Before function call: var1 is $var1 : var2 is $var2

var_change

echo After function call: var1 is $var1 : var2 is $var2
```



```
user@bash: ./local_variables.sh
Before function call: var1 is global 1 : var2 is global 2
Inside function: var1 is local 1 : var2 is global 2
```

Example output



```
user@bash: ./local_variables.sh
Before function call: var1 is global 1 : var2 is global 2
Inside function: var1 is local 1 : var2 is global 2
After function call: var1 is global 1 : var2 is 2 changed again
user@bash:
```

Built-in Functions

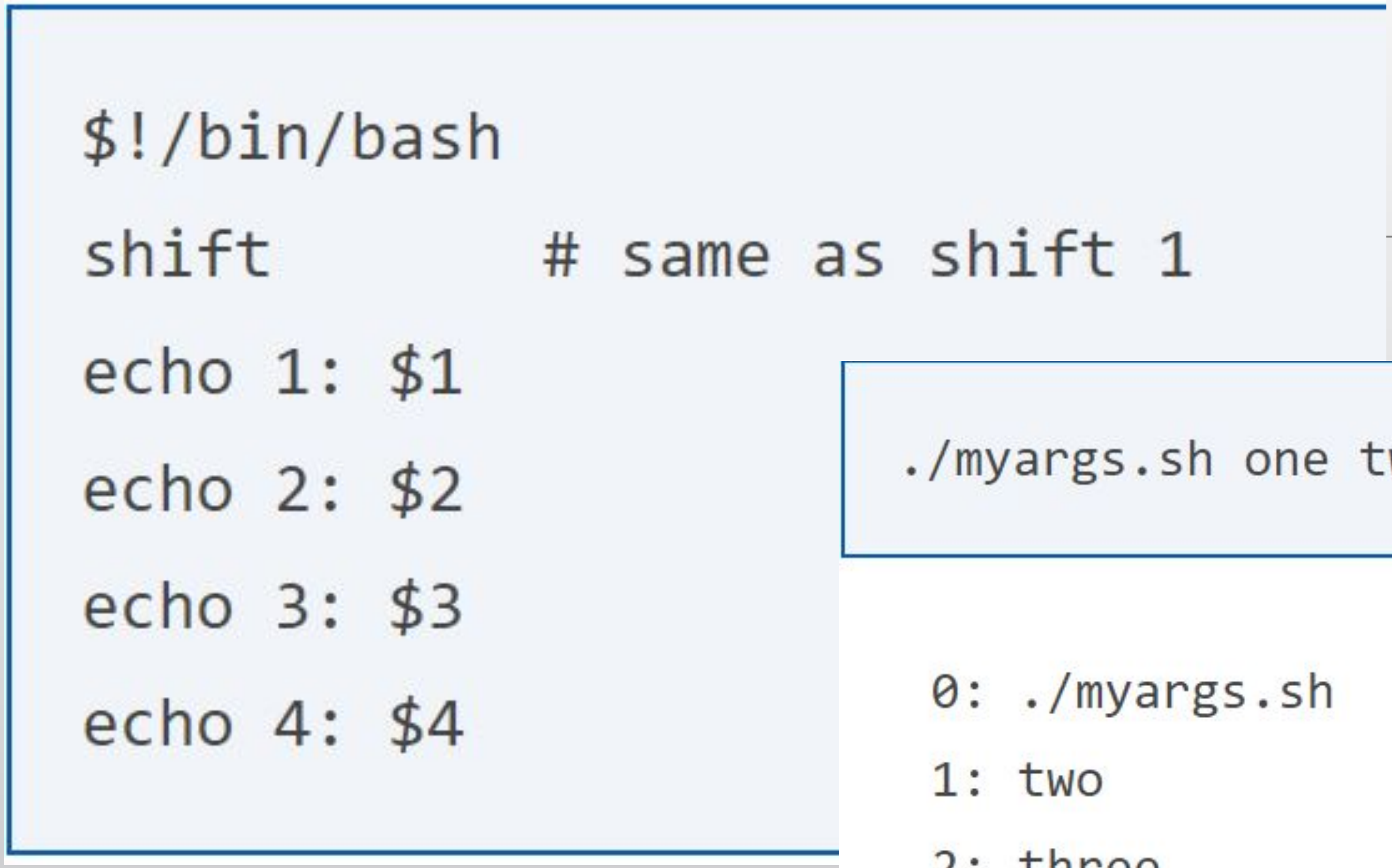
- **Built-in functions**
  - Built- in functions are the function already defined in a programming language
  - the program does not require defining it
  - they directly use it with in their code
  - Some built-in functions are also defined in shell scripting e.g. random function etc.

Example:

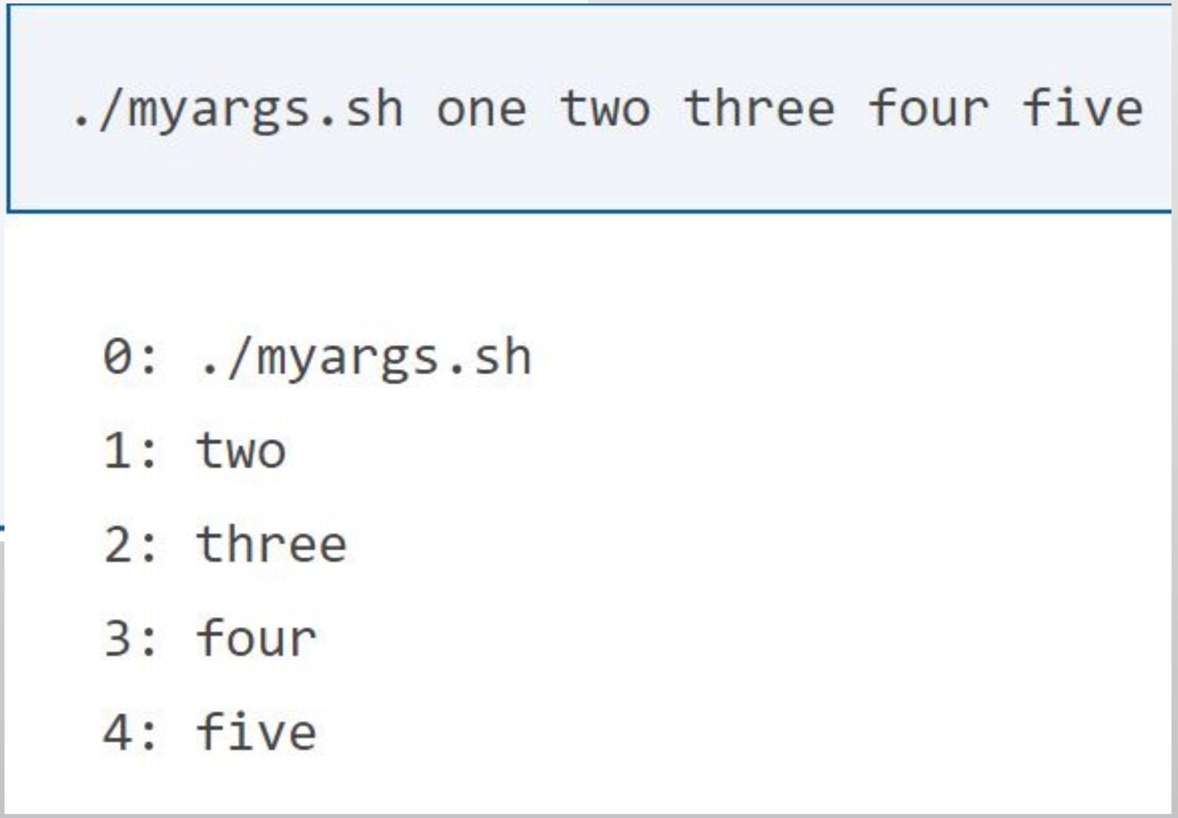
```
num_rows=4
for ((i=1;i<=num_rows;i++))
 do
       echo $RANDOM
done
```

Built-in Functions

- **Built-in functions**
  - Shift - shift [*n*] (Shift the positional parameters to the left by *n*)
  - Test- test *expr* (evaluate a conditional expression and return status of 0/1)



```
$!/bin/bash
shift       # same as shift 1
echo 1: $1
echo 2: $2
echo 3: $3
echo 4: $4
```



```
./myargs.sh one two three four five
```

```
0: ./myargs.sh
1: two
2: three
3: four
4: five
```

Recursive functions

- ❑A recursive function is a function that calls itself during its execution.
- ❑This enables the function to repeat itself several times
  - ❑ outputting the result at the end of each iteration

Recursive functions

- You can write a recursive function in shell script
- Example

```
array=(1 2 3 4)
f() {
        if [ "$1 == -1 ]
       then
                return
       else
                a=$1
                echo $((array[$a]))  //2,1,
                f $((a-1)) #recursive function call
         fi  }
f 1 #function call with parameter
```