**Metacharacter**

[Linux shell programming : metacharacters & quotes - 2020 (bogotobogo.com)](https://www.bogotobogo.com/Linux/linux_shell_programming_tutorial7_metacharacters_quotes.php)

[quoting - When to use bash ANSI C style escape, e.g. $'\n' - Unix & Linux Stack Exchange](https://unix.stackexchange.com/questions/155367/when-to-use-bash-ansi-c-style-escape-e-g-n)

Sometimes we need to pass metacharacters to the command being run and do not want the shell to interpret them. There are three options to avoid shell interpretation of metacharacters.

* Escape the metacharacter with a backslash (\). (See also Escaped Charaters) Escaping characters can be inconvenient to use when the command line contains several metacharacters that need to be escaped.
* Use single quotes (' ') around a string. Single quotes protect all characters except the backslash (\).
* Use double quotes (" "). Double quotes protect all characters except the backslash (\), dollar sign ($) and grave accent (`). Double quotes is often the easiest to use because we often want environment variables to be expanded.

#Declare bash string variable

BASH\_VAR="Bash Script"

# echo variable BASH\_VAR

echo $BASH\_VAR

#when meta character such us "$" is escaped with "\" it will be read literally

echo \$BASH\_VAR

# backslash has also special meaning and it can be suppressed with yet another "\"

echo "\\"

#Declare bash string variable

BASH\_VAR="Bash Script"

# echo variable BASH\_VAR

echo $BASH\_VAR # --> Bash Script

# meta characters special meaning in bash is suppressed when  using single quotes

echo '$BASH\_VAR  "$BASH\_VAR"' # --> $BASH\_VAR  "$BASH\_VAR"

#It is not possible to use another single quote within two single quotes not even if the single quote is escaped by backslash.

echo "$BASH\_VAR  '$BASH\_VAR'" # --> Bash Script  'Bash Script'

#Declare bash string variable

BASH\_VAR="Bash Script"

# echo variable BASH\_VAR

echo $BASH\_VAR

# meta characters and its special meaning in bash is suppressed when using double quotes except "$", "\" and "`"

echo "It's $BASH\_VAR  and \"$BASH\_VAR\" using backticks: `date`"

#Declare bash string variable

BASH\_VAR="Bash Script"

# as a example we have used \n as a new line, \x40 is hex value for @

# and  is octal value for .

echo $'web: www.linuxconfig.org\nemail: web\x40linuxconfigorg$BASH\_VAR'

# web: www.linuxconfig.org

#email: web@linuxconfigorg$BASH\_VAR

echo $(pwd) # --> /d/shell/shell/basic

echo `pwd` # --> /d/shell/shell/basic

echo pwd # --> pwd

echo 'pwd' # --> pwd

echo "pwd"  # --> pwd

# echo variable BASH\_VAR

echo $BASH\_VAR

echo $'$BASH\_VAR\nok'

echo 'a\nb' #--> a\nb

echo $'a\nb'

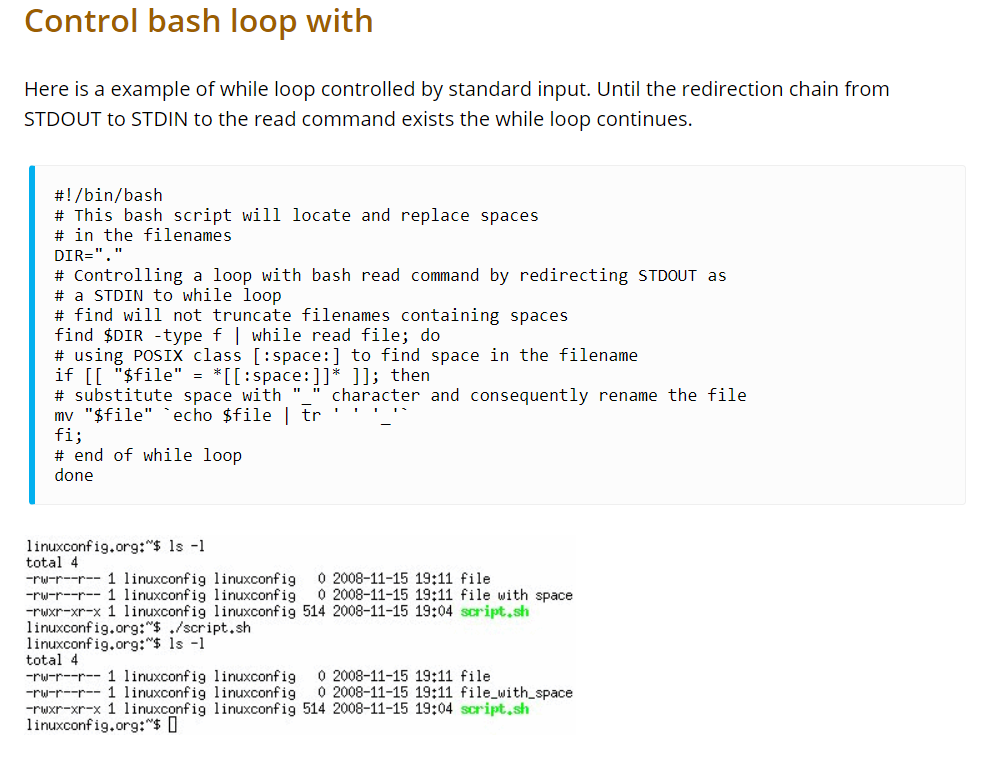
# --> a

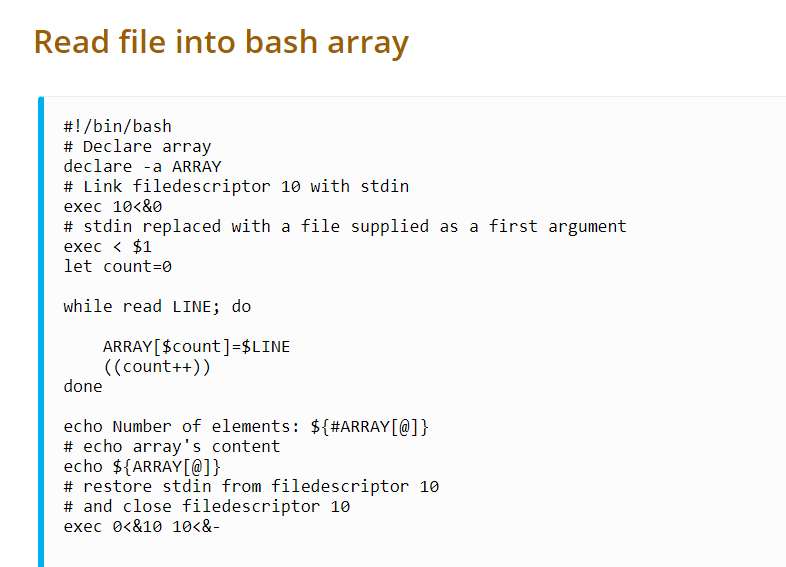
# -->b

echo "a\nb" # --> a\nb

**Find**

[find command in Linux with examples - GeeksforGeeks](https://www.geeksforgeeks.org/find-command-in-linux-with-examples/)

****

****

**Declare**

[Typing variables: declare or typeset (tldp.org)](https://tldp.org/LDP/abs/html/declareref.html#FTN.AEN5685)

[Bash declare command – Linux Hint](https://linuxhint.com/bash_declare_command/)

* Using the declare built-in restricts the scope of a variable
* In this context, typing a variable means to classify it and restrict its properties. For example, a variable declared or typed as an integer is no longer available for string operations.
* **declare -i intvar**

**intvar=23**

**echo "$intvar" # 23**

**intvar=stringval**

**echo "$intvar" # 0**

#!/bin/bash

declare -r var1=1

echo "var1 = $var1"

(( var1++ )) # var1: readonly variable

declare -i number

# The script will treat subsequent occurrences of "number" as an integer.

number=3

echo "Number = $number"     # Number = 3

number=three

echo "Number = $number"     # Number = 0

# Tries to evaluate the string "three" as an integer.

n=6/3

echo "n = $n"       # n = 6/3

declare -i n

n=6/3

echo "n = $n"       # n = 2

declare -i var1   # var1 is an integer.

var1=2367

echo "var1 declared as $var1"

var1=var1+1       # Integer declaration eliminates the need for 'let'.

echo "var1 incremented by 1 is $var1."

# Attempt to change variable declared as integer.

echo "Attempting to change var1 to floating point value, 2367.1."

var1=2367.1       # Results in error message, with no change to variable.

echo "var1 is still $var1"

declare -a indices

#The variable indices will be treated as an array.

#A declare -f line with no arguments in a script causes a listing of all the functions previously defined in that script.

declare -f function\_name

#A declare -f function\_name in a script lists just the function named.

func1 ()

{

  echo This is a function.

}

declare -f        # Lists the function above.

declare -x var3=373

#The declare command permits assigning a value to a variable in the same statement as setting its properties.

**Bash trap command**

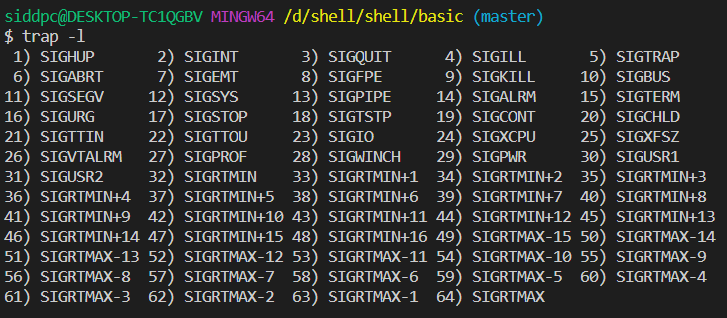
[Bash trap command – Linux Hint](https://linuxhint.com/bash_trap_command/)

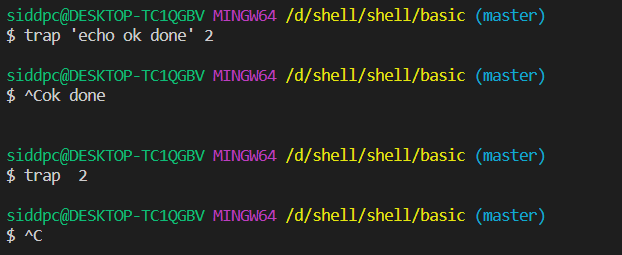
* A built-in bash command that is used to execute a command when the shell receives any signal is called `trap`. When any event occurs then bash sends the notification by any signal.
* Many signals are available in bash. The most common signal of bash is SIGINT (Signal Interrupt). When the user presses CTRL+C to interrupt any process from the terminal then this signal is sent to notify the system.
* SIGTERM signal is used to terminate the process immediately by releasing its resources.
* **trap [action] [signal]**
* **$ trap 'rm temp.txt' err exit**

**$ ls**

**$ exit**

|  |  |
| --- | --- |
| **Key** | **Description** |
| -l | It is used to display the list of all signal names with corresponding number. |
| -p | It is used to display signal command or trap command for signal\_spec. |
| arg | It is used to execute a command when the shell receives the signal(s). |
| signal\_spec | It contains signal name or signal number. |

The signal number of **SIGUP, SIGQUIT and SIGKILL are 1, 3 and 9**. The following first command will set a trap for these three signals. When any of these signals will occur then the message “Trap command is executed” will print. Run the following command from the terminal.

* **$ trap 'echo Trap command executed' 1 3 9**
* When the user will press Ctrl+C to generate the signal assign by `trap` command then the `echo` command of trap command will execute and the following output will appear.

# Call func function on exit

trap func exit

# Declare the function

function func() {

  echo "Task completed"

}

# Read the files and folders of the current directory list using for loop

for i  in \*

do

  echo "$i"

done