**Basics about shell scripting**

**if loop**

**if condition** which is used for decision making in shell script, If given condition is true then command1 is executed.  
*Syntax:*

*if condition*

*then*

*command1 if condition is true or if exit status*

*of condition is 0 (zero)*

*...*

*...*

*fi*

**Example 1**

$ ./scriptname showfile

#!/bin/sh

#Script to check whether input is a file or not

if [[ -f $1 ]]

then

echo -e "\n\n $1 is a file"

fi

**O/P:** showfile is a file

**Example 2: Print 10 number**

#!/bin/sh

# Script to see whether argument is positive

a=1

if $a -le 10

then

echo "$a\n"

fi

**Mathematical operators in shell**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mathematical Operator in  Shell Script** | **Meaning** | **Normal Arithmetical/ Mathematical Statements** | **But in Shell** | |
|  |  |  | **For test statement with if command** | **For [ expr ] statement with if command** |
| -eq | is equal to | 5 == 6 | if test 5 -eq 6 | if [ 5 -eq 6 ] |
| -ne | is not equal to | 5 != 6 | if test 5 -ne 6 | if [ 5 -ne 6 ] |
| -lt | is less than | 5 < 6 | if test 5 -lt 6 | if [ 5 -lt 6 ] |
| -le | is less than or equal to | 5 <= 6 | if test 5 -le 6 | if [ 5 -le 6 ] |
| -gt | is greater than | 5 > 6 | if test 5 -gt 6 | if [ 5 -gt 6 ] |
| -ge | is greater than or equal to | 5 >= 6 | if test 5 -ge 6 | if [ 5 -ge 6 ] |

**String comparision use.**

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| string1 = string2 | string1 is equal to string2 |
| string1 != string2 | string1 is NOT equal to string2 |
| string1 | string1 is NOT NULL or not defined |
| -n string1 | string1 is NOT NULL and does exist |
| -z string1 | string1 is NULL and does exist |

**Test for file and directory types**

|  |  |
| --- | --- |
| **Test** | **Meaning** |
| -s file | Non empty file |
| -f file | Is File exist or normal file and not a directory |
| -d dir | Is Directory exist and not a file |
| -w file | Is writeable file |
| -r file | Is read-only file |
| -x file | Is file is executable |

**if...else...fi**

If given condition is true then command1 is executed otherwise command2 is executed.

Syntax:

if condition

then

condition is zero (true - 0)

execute all commands up to else statement

else

if condition is not true then

execute all commands up to fi

fi

**Nested if-else-fi**

Syntax:

if condition

then

if condition

then

.....

..

do this

else

....

..

do this

fi

else

...

.....

do this

fi

**Multilevel if-then-else**

**Syntax:**

if condition

then

condition is zero (true - 0)

execute all commands up to elif statement

elif condition1

then

condition1 is zero (true - 0)

execute all commands up to elif statement

elif condition2

then

condition2 is zero (true - 0)

execute all commands up to elif statement

else

None of the above condtion,condtion1,condtion2 are true (i.e.

all of the above nonzero or false)

execute all commands up to fi

fi

**for Loop**

***Syntax:***

for { variable name } in { list }

do

execute one for each item in the list until the list is

not finished (And repeat all statement between do and done)

done

***Example:***

for a in 1 2 3 4 6

do

echo $a

done

**OR**

for (( a=1; a<=6; a++ ))

do

echo $a

done

**Output**

1

2

..

6

**while loop**

Syntax:

while [ condition ]

do

command1

command2

command3

..

Increment of variable

Done

Example:

i=1

while [ i –ge 5 ]

do

echo $i

((i++))

done

Output

1

2

…

5

**Until loop**

**Syntax:**

until [ condition ]

do

command1

command2

...

....

increment of variable

done

**Example**

a=1

until [ a <= 5]

do

echo $a

((a++))

done

**Output:**

1

2

3

4

5

**Break and continuous:**

**Break :** Use the break statement to exit from within a FOR, WHILE or UNTIL loop i.e. stop loop execution.

#!/bin/bash

a=1

while [ $a -le 5 ]

do

echo $a

((a++))

[[ $a -eq 4 ]] && break

done

Output:

1

2

..

5

**Continuous**: The continue statement is used to resume the next iteration of the enclosing FOR, WHILE or UNTIL loop.

#!/bin/bash

for i in 1 2 3 4 5 6

do

### just skip printing $i; if it is 3 or 6 ###

if [ $i -eq 3 -o $i -eq 6 ]

then

continue ### resumes iteration of an enclosing for loop ###

fi

# print $i

echo "$i"

done

***Output***:

1

2

4

5

**Case statement**

The case construct in bash shell allows us to test strings against patterns that can contain wild card characters.

case expression in

pattern1 )

statements ;;

pattern2 )

statements ;;

...

esac

Example:

if [ -z $1 ]

then

rental="\*\*\* Unknown vehicle \*\*\*"

elif [ -n $1 ]

then

# otherwise make first arg as rental

rental=$1

fi

case $rental in

"car") echo "For $rental Rs.20 per k/m";;

"van") echo "For $rental Rs.10 per k/m";;

"jeep") echo "For $rental Rs.5 per k/m";;

"bicycle") echo "For $rental 20 paisa per k/m";;

\*) echo "Sorry, I can not gat a $rental for you";;

esac

Exection: ./scriptname.sh car

Output: For car Rs.20 per k/m

**Debug Scripts:**

While programming shell sometimes you need to find the errors (bugs) in shell script and correct the errors (remove errors - debug). For this purpose you can use -v and -x option with sh or bash command to debug the shell script. General syntax is as follows:

*Syntax:*  
sh   option   { shell-script-name }  
**OR**  
bash   option   { shell-script-name }  
Option can be  
**-v**Print shell input lines as they are read.  
**-x**After expanding each simple-command, bash displays the expanded value of PS4 system variable, followed by the command and its expanded arguments.

*Example: bash –v scriptname.sh OR sh –x scriptname.sh*

**Functions:**

When program gets complex we need to use divide and conquer technique. It means whenever programs gets complicated, we divide it into small chunks/entities which is know as function.

Function is series of instruction/commands. Function performs particular activity in shell i.e. it had specific work to do or simply say task. To define function use following syntax:

Syntax:

function-name ( )

{

command1

command2

.....

...

commandN

return

}

Where function-name is name of you function, that executes series of commands. A return statement will terminate the function. Example:

Type Hello() at $ prompt as follows

$ Hello()

{

echo "Hello $LOGNAME, Have nice scripting"

return

}

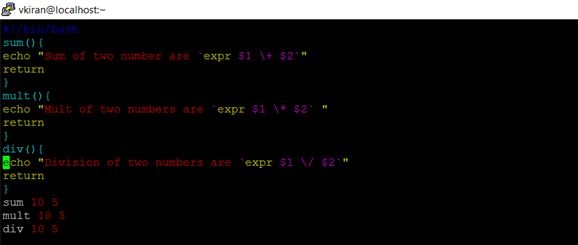
To execute this Hello() function just type it name as follows:

$ Hello

Output:

Hello Raj, Have nice scripting.

Few more example for functions:



Example for return statement:

add()

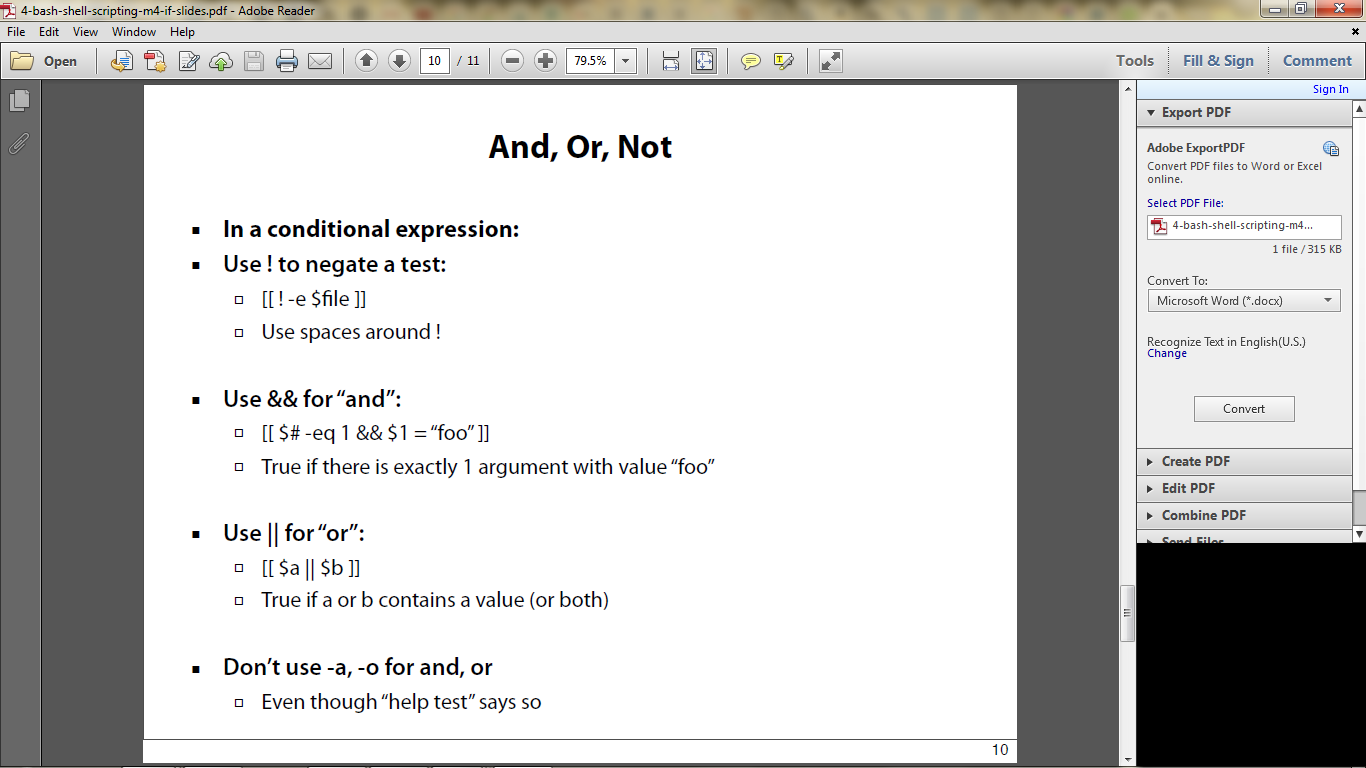
{

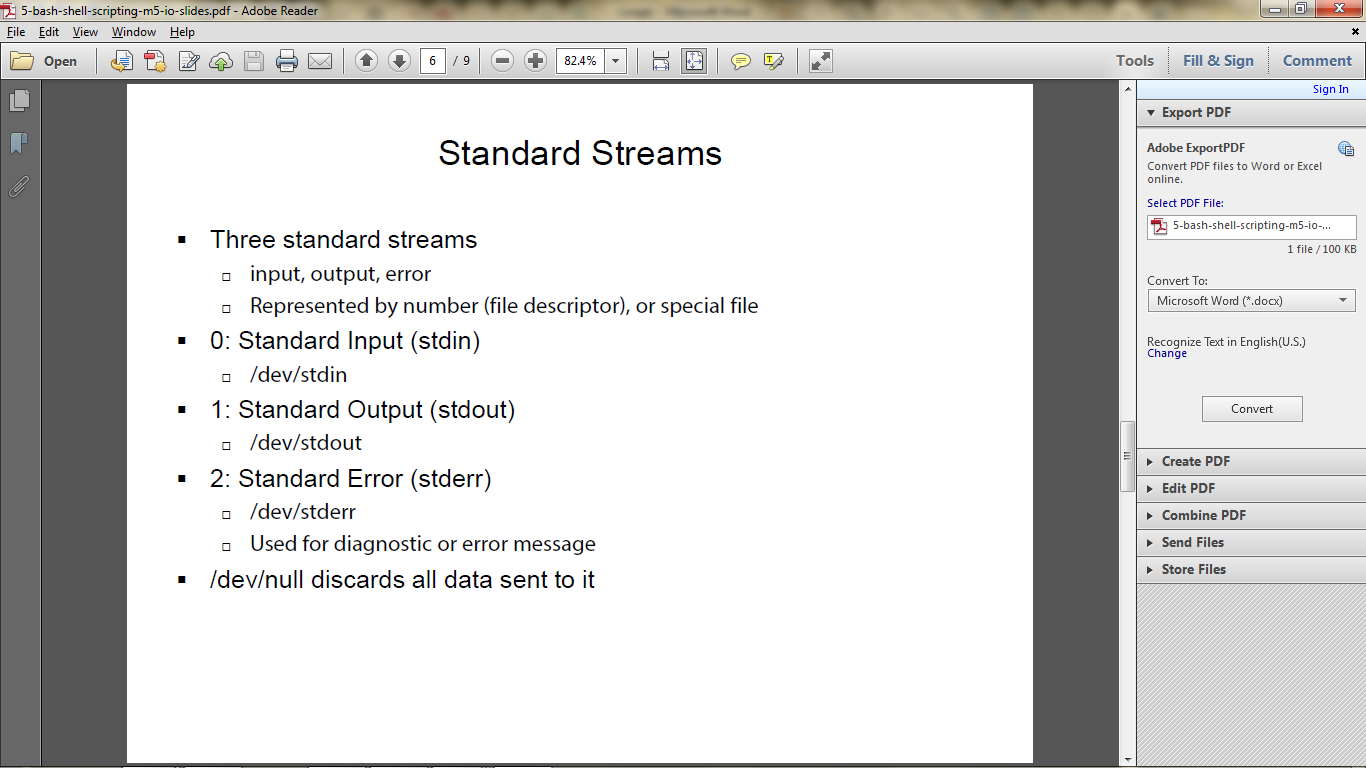
return $(expr "$1" + "$2")

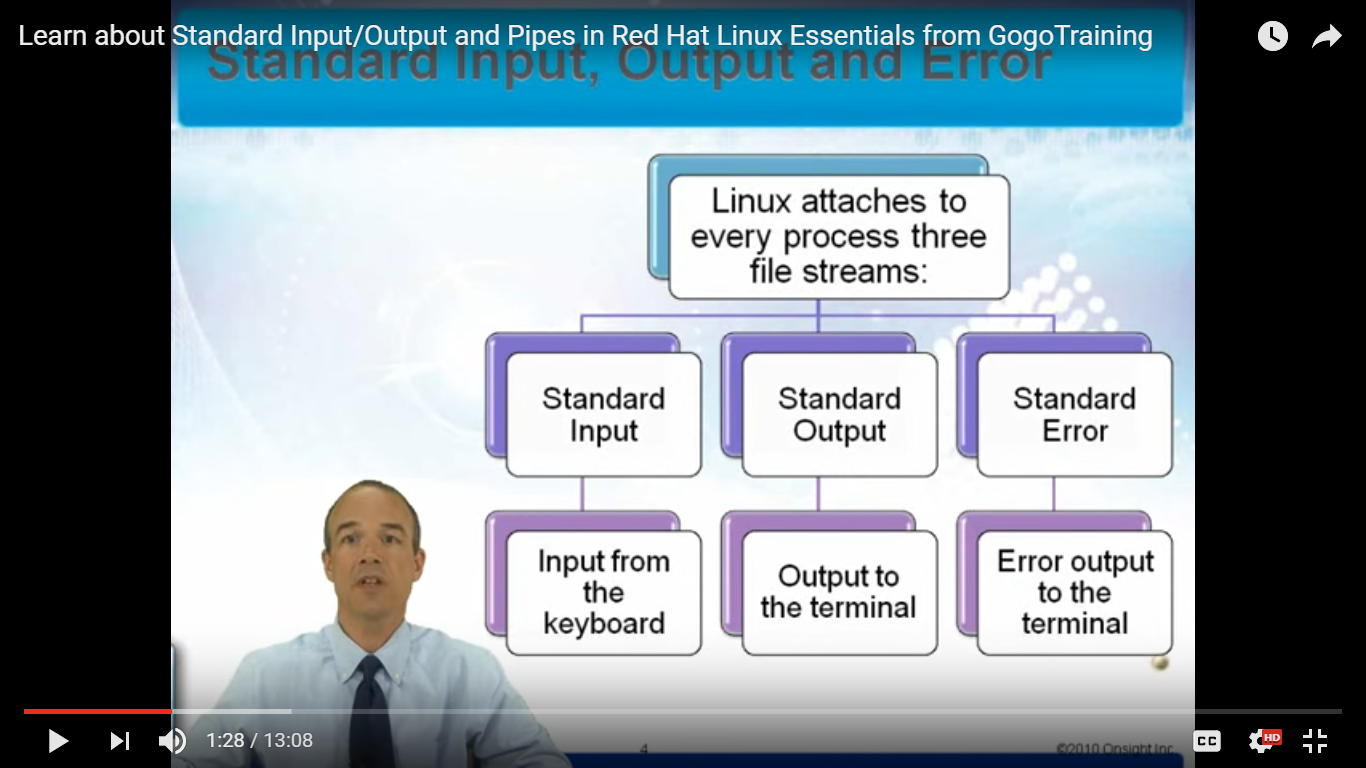
}

add $1 $2

echo $?







Standard input

Example:

wc

--🡪 It waits for input

wc

1 2 4 5

clt+d

0 4 7 (lines words characters)

Standard Output

ls

ls > file.out

ls > /dev/null # If you don’t want to save the output

Standard error

lss

lss > /dev/null

lss 2> stdout.out

Stander

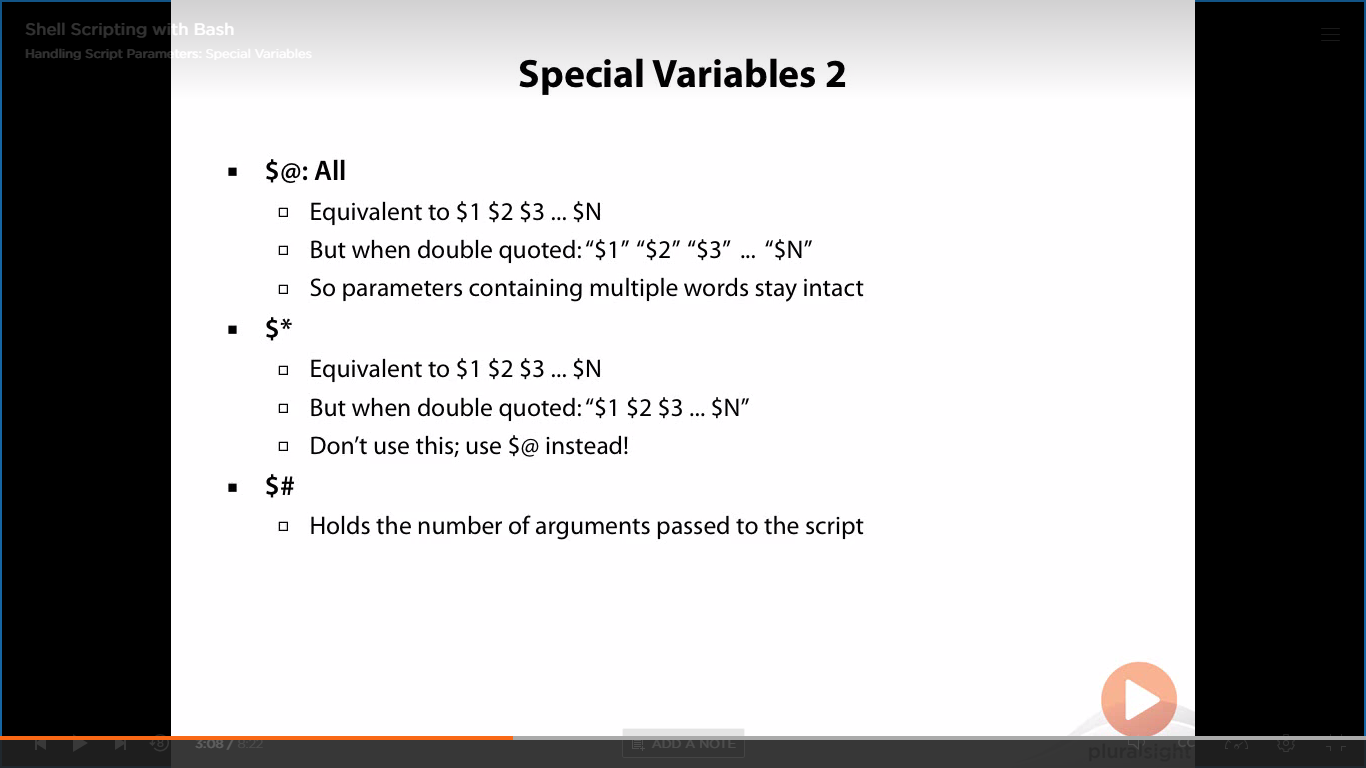
ls && lss

We will get both output and erro

ls /usr/local /binn > errout.out 🡪 It will redirect the output to the file and error will be printed

if you want to redirector both

ls /usr/local /binn > errout.out 2>&1

  
Example

#!/bin/bash

#echo "print $@"

#echo "print $\*"

#echo "print $#"

echo Test for $@

for a in "$@" -🡪 Simply using $@ will not solve the problem, but you need to use “$@” and then it works.

do

echo "$a"

done

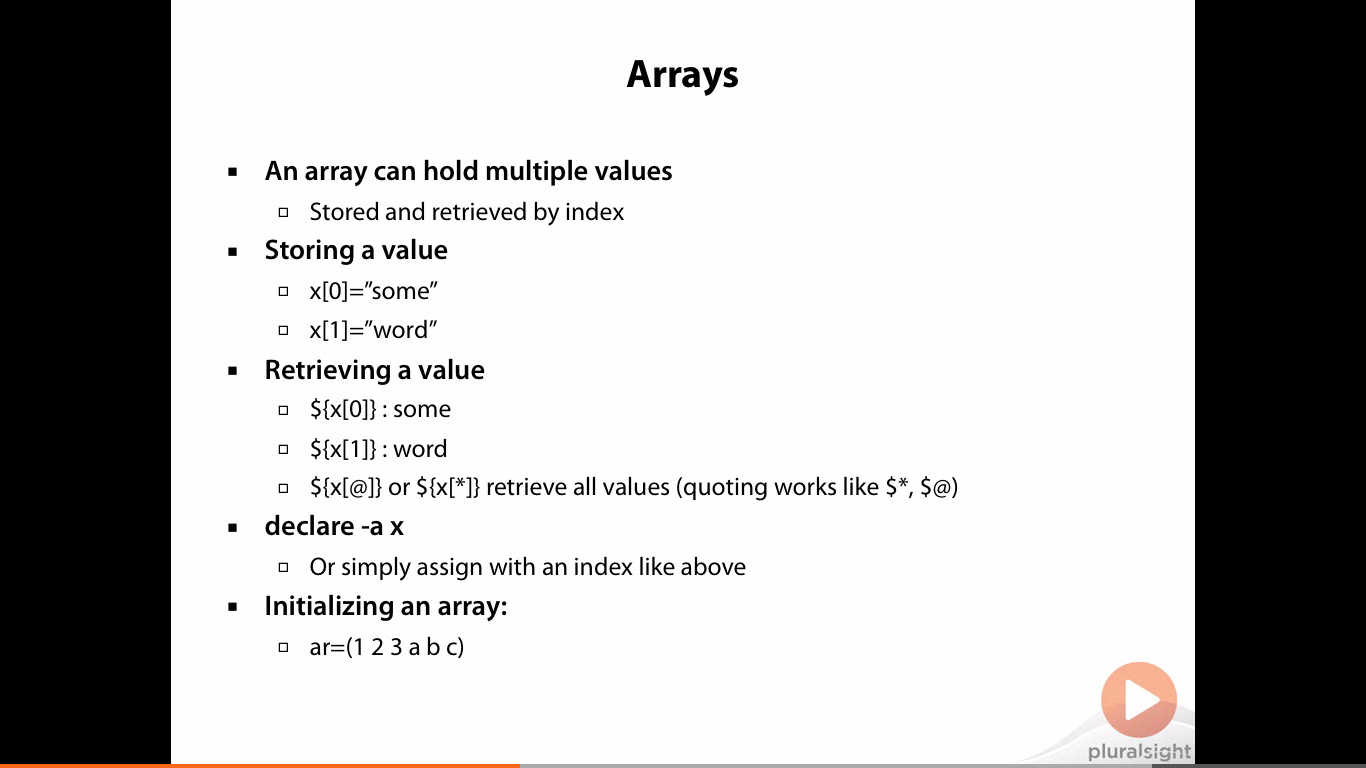
echo Test for $\*

for b in $\*

do

echo "$b"

done



a=(1 2 3 4 5)

echo $a[0]

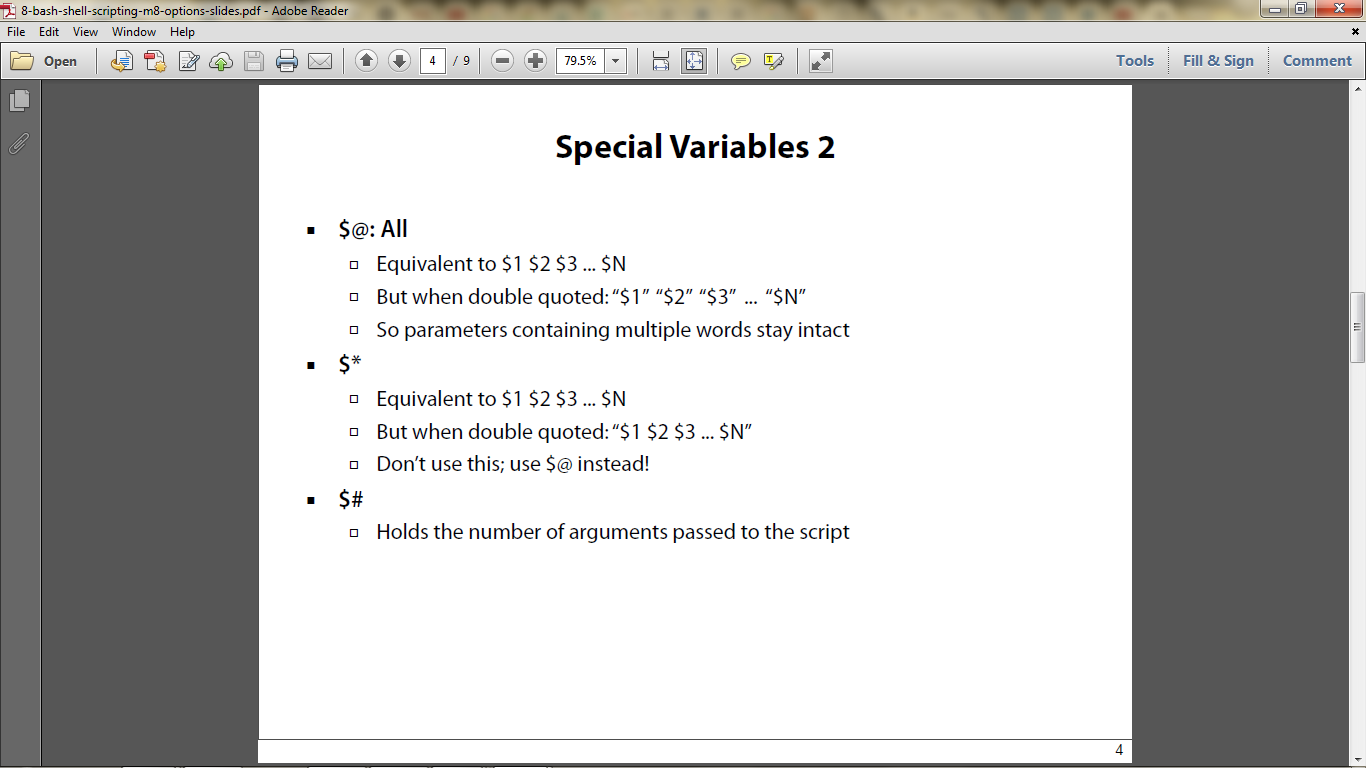
All element in array

echo ${a[@]}

echo ${a[\*]}

Length of array

echo ${#a[@]}



**DNS Syntax Types Explained**

An “A” record, which stands for “address” is the most basic type of syntax used in DNS records, indicating the actual IP address of the domain. The “AAAA” record is an IPV6 address record that maps a hostname to a 128-bit Ipv6 address.  Regular DNS addresses are mapped for 32-bit IPv4 addresses.

The “CNAME” record stands for “canonical name” and serves to make one domain an alias of another domain. CNAME is often used to associate new subdomains with an existing domain's DNS records.

The “MX” record stands for “mail exchange” and is basically a list of mail exchange servers that are to be used for the domain.

The “PTR” record stands for “pointer record” and maps an Ipv4 address to the CNAME on the host.

The “NS” record stands for “name server” and indicates which Name Server is authoritative for the domain.

An “SOA” record stands for “State of Authority” and is easily one of the most essential DSN records because it stores important information like when the domain was last updated and much more.

An “SRV” record stands for “service” and is used to define a TCP service on which the domain operates.

A “TXT” record lets the administrator insert any text they'd like into the DNS record, and it is often used for denoting facts about the domain.

IP Address

http://www.bleepingcomputer.com/tutorials/ip-addresses-explained/