SHELL SCRIPTING

Shilpa sunil
Paul ajith
Sandra maria jose

February 21, 2017

Table of contents

- SHELL PARAMETERS
  - POSITIONAL PARAMETERS
  - SPECIAL PARAMETER
- IFS
- CONDITIONAL COMMANDS
- SHELL BASIC OPERATIONS

SHELL PARAMETERS

A parameter is an entity that stores values. It can be a name, a number, or one of the special characters listed below.

- POSITIONAL PARAMETERS:The shell's command-line arguments.
- SPECIAL PARAMETERS:Parameters denoted by special characters.

POSITIONAL PARAMETERS

A positional parameter is a parameter denoted by one or more digits, other than the single digit 0.
Positional parameters are assigned from the shell's arguments when it is invoked, and may be reassigned using the set builtin command.
Positional parameter N may be referenced as $\$N$, or as $\$N$when N consists of a single digit. Positional parameters may not be assigned to with assignment statements. The set and shift builtins are used to set and unset them
The positional parameters are temporarily replaced when a shell function is executed
When a positional parameter consisting of more than a single digit is expanded, it must be enclosed in braces.

SPECIAL PARAMETERS

The shell treats several parameters specially. These parameters may only be referenced; assignment to them is not allowed.

- $*:
  - When the expansion is not within double quotes, each positional parameter expands to a separate word. In contexts where it is performed, those words are subject to further word splitting and pathname expansion.
  - When the expansion occurs within double quotes, it expands to a single word with the value of each parameter separated by the first character of the IFS special variable.
- $@:
  - When the expansion occurs within double quotes, each parameter expands to a separate word.
  - If the double-quoted expansion occurs within a word, the expansion of the first parameter is joined with the beginning part of the original word, and the expansion of the last parameter is joined with the last part of the original word.
  - When there are no positional parameters, "@" *and* @ expand to nothing (i.e., they are removed).

SPECIAL PARAMETERS.

- $:Expands to the number of positional parameters in decimal.
- $?:Expands to the exit status of the most recently executed foreground pipeline.
- $−:Expands to the current option flags as specified upon invocation, by the set builtin command, or those set by the shell itself (such as the -i option).
- $$:Expands to the process ID of the shell. In a () subshell, it expands to the process ID of the invoking shell, not the subshell.
- $!:Expands to the process ID of the job most recently placed into the background, whether executed as an asynchronous command or using the bg builtin

IFS

- IFS stands for **"internal field separator"**. It is used by the shell to determine how to do word splitting, i. e. how to recognize word boundaries.
- The IFS is a special shell variable.
- You can change the value of IFS as per your requirments.
- The Internal Field Separator (IFS) that is used for word splitting after expansion and to split lines into words with the read builtin command.
- The default value is <space><tab><newline>.

COMPOUND COMMANDS

- Looping Constructs:Shell commands for iterative action.
- Conditional Constructs:Shell commands for conditional execution.

LOOPING CONSTRUCTS

- until:
  The syntax of the until command is:
  until test-commands; do consequent-commands; done/
- while
  The syntax of the while command is:
  while test-commands; do consequent-commands; done
- for:
  The syntax of the for command is:
  for name [[in [words . . .]] ;] do commands; done

CONDITIONAL CONSTRUCTS

- if
  The syntax of the if command is:
  if test-commands; then
  consequent-commands;
  elif more-test-commands; then
  more-consequents;
  else alternate-consequents;
  fi
- case The syntax of the case command is:
  case word in [[(] pattern [| pattern]. . .) command-list ;;]. . .
  esac
- select
  It has almost the same syntax as the for command:
  select name [in words . . .]; do commands; done

SHELL BASIC OPERATORS

- Arithmetic Operators
- Relational Operators
- Boolean Operators
- String Operators
- File Test Operators

ARITHMETIC OPERATORS

Assume variable a holds 10 and variable b holds 20 then

OPERATORS	EXAMPLE
ADDITION	‘expr $b + $a“ will give 30
SUBTRACTION	‘expr $b -$a‘ will give -10
MULTIPLICATION	‘expr‘$b $a‘ will give 200
DIVISION	‘expr $b / $a‘ will give 2
MODULUS	‘expr $b modulu $a‘ will give 0
ASSIGNMENT	a = $b would assign value of b into a
EQUALITY	[$a == $b] would return false.
NOTEQUALITY	[$a!= $b] would return true.

BOOLEAN OPERATORS

Assume variable a holds 10 and variable b holds 20 then

OPERATORS	EXAMPLE
!	[! false] is true.
-o	[$*a*-lt 20 -o $*b* -gt 100] is true.
. -a	[$*a* -lt 20 -a $*b*-gt 100] is false.

STRING OPERATORS

Assume variable a holds "abc" and variable b holds "efg" then

OPERATORS	EXAMPLE
=	[$a = $b -gt 100]is not true
!=	[$a!= $b -gt 100] is true.
. -z	[-z $a] is not true.
-n	[-n $a] is true.
str	[$a] is not false. if str is not the empty string;

FILE TEST OPERATORS

Assume a variable file holds an existing file name "test" the size of which is 100 bytes and has read, write and execute permission on

OPERATORS	EXAMPLE
-b file	[-b $*file*]is false
-c file	[-c $*file*]is false
-d file	[-d $*file*]is false
-f file	[-f $*file*]is true
-g file	[-g $*file*]is false

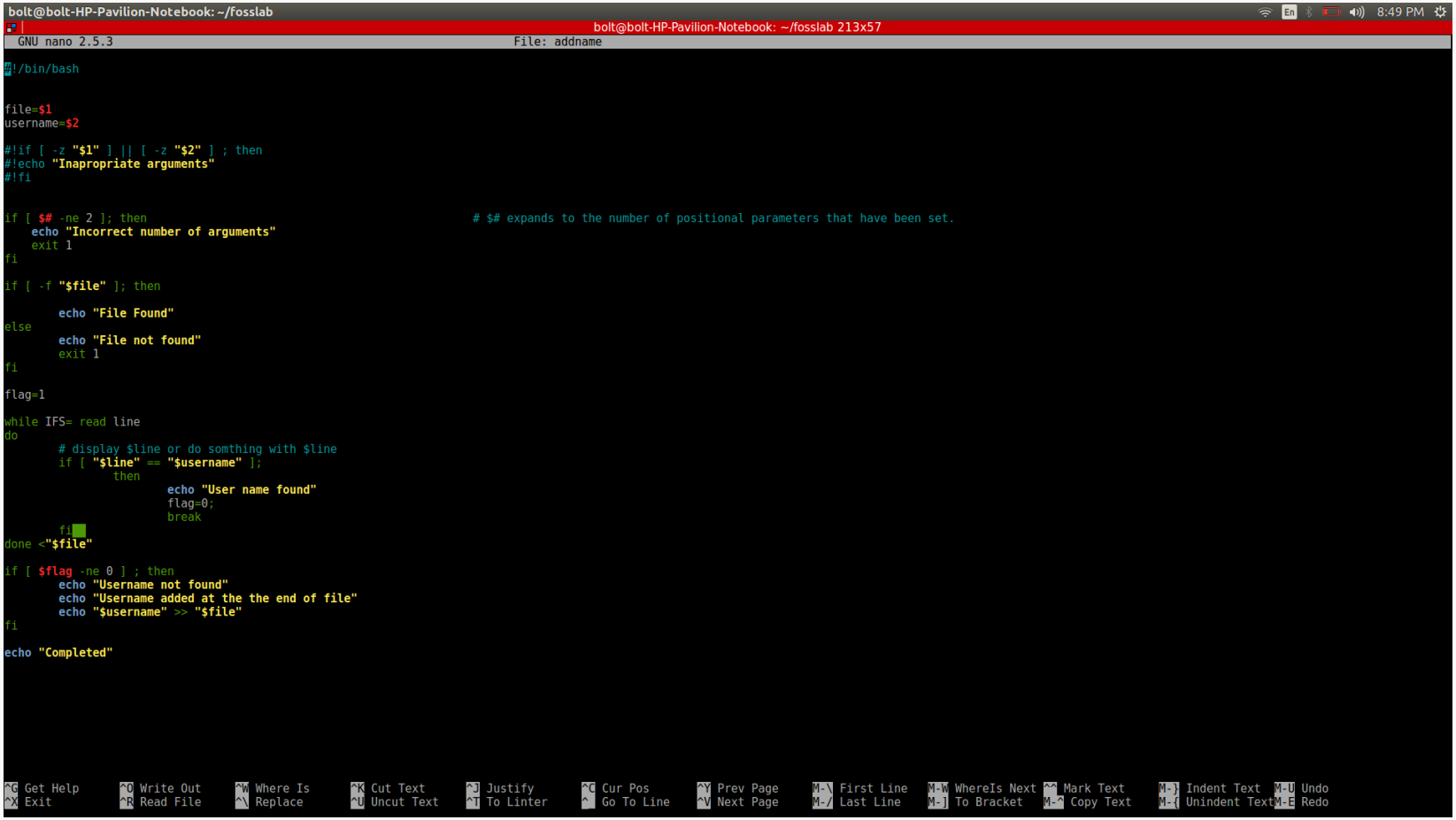

Figure: program code