Bash Shell Script CodeCount™

Counting Standard

University of Southern California

Center for Systems and Software Engineering

May , 2010

Revision Sheet

Date	Version	Revision Description	Author
5/11/2010	1.0	Original Release	CSSE
1/2/2013	1.1	Updated document template	CSSE

Table of Contents

1. Definitions

1.1. **SLOC –** Source Lines of Code is a unit used to measure the size of software program. SLOC counts the program source code based on a certain set of rules. SLOC is a key input for estimating project effort and is also used to calculate productivity and other measurements.

1.2. **Physical SLOC –** One physical SLOC is corresponding to one line starting with the first character and ending by a carriage return or an end-of-file marker of the same line, and which excludes the blank and comment line.

1.3. **Logical SLOC –** Lines of code intended to measure "statements", which normally terminate by a semicolon (C/C++, Java, C#) or a carriage return (VB, Assembly), etc. Logical SLOC are not sensitive to format and style conventions, but they are language-dependent.

1.4. **Data declaration line or data line –** A line that contains declaration of data and used by an assembler or compiler to interpret other elements of the program.

The following table lists the Bash Shell Script keywords that denote data declaration lines:

declare
local
type
typeset

Table 1 Data Declaration Types

1.5. **Compiler Directives –** A statement that tells the compiler how to compile a program, but not what to compile. Bash Shell Script does not contain any compiler directives.

1.6. **Blank Line –** A physical line of code, which contains any number of white space characters (spaces, tabs, form feed, carriage return, line feed, or their derivatives).

1.7. **Comment Line –** A comment is defined as a string of zero or more characters that follow language-specific comment delimiter.

Bash Shell Script comment delimiter is "#". A whole comment line may span one line and does not contain any compliable source code. An embedded comment can co-exist with compliable source code on the same physical line. Banners and empty comments are treated as types of comments.

1.8. **Executable Line of code –** A line that contains software instruction executed during runtime and on which a breakpoint can be set in a debugging tool. An instruction can be stated in a simple or compound form.

- An executable line of code may contain the following program control statements:
  - Selection statements (if, select, case)
  - Iteration statements (for, while, until)
  - Empty statements (one or more ";")
  - Jump statements (return, break, continue, exit)
  - Expression statements (function calls, assignment statements, operations, etc.)
  - Block statements
- An executable line of code may not contain the following statements:
  - Data declaration (data) lines
  - Whole line comments, including empty comments and banners
  - Blank lines

2. Checklist for source statement counts

PHYSICAL SLOC COUNTING RULES

MEASUREMENT UNIT	ORDER OF PRECEDENCE	PHYSICAL SLOC	COMMENTS
Executable Lines	1	One per line	Defined in 1.8
Non-executable Lines			
Declaration (Data) Lines	2	One per line	Defined in 1.4
Compiler Directives	3	NA	Defined in 1.5
Comments			Defined in 1.7
On their own lines	4	Not Included	
Embedded	5	Not Included	
Banners	6	Not Included	
Empty Comments	7	Not Included	
Blank lines	8	Not Included	Defined in 1.6

LOGICAL SLOC COUNTING RULES

NO.	STRUCTURE	ORDER OF PRECEDENCE	LOGICAL SLOC RULES	COMMENTS
R01	“for”,”while” or “if” statement	1	Count once	“while” is an independent statement
R02	“do{} while()” statement	2	Count once	Braces {..} or semicolon ; used with statement are not counted
R03	Statements ending with a semicolon or new line	3	Count once per statement, including empty statement	Semicolons within “for” statements are not counted. Semicolons used with R01 and R02 are not counted.
R04	Block delimiters, braces {..}	4	Count once per pair of braces {..}, except where a closing brace is followed by a semicolon, i.e. }; or an opening brace comes after a keyword “else”.	Braces used with R01 and R02 are not counted. Function definition is counted once since it is followed by {..}.
R05	Compiler directives	5	NA	No compiler directives for bash

3. Examples

EXECUTABLE LINES

SELECTION Statement

ESS1 – if, elif, else and nested if statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

if *test-commands*; then
consequent-commands;
[elif *more-test-commands*; then
more-consequents;]
[else *alternate-consequents*;]
fi
	if [$A == foo];
	then echo oof;
	elif [$A == bar];
	then echo rab;
	elif [$A == baz];
	then echo zab;
	else
	echo nile;
	fi
		1
		1
		1
		1
		1
		1
		0
		1
		0

ESS2 – case and nested case statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
case *word* in [[(] *pattern* [| *pattern*]...) *command-list* ;;]... esac	echo -n "Enter an animal name: " read ANIMAL echo -n "The $ANIMAL has " case $ANIMAL in horse | dog | cat) echo -n "four";; man | kangaroo) echo -n "two";; *) echo -n "unknown";; esac echo " legs."	1 1 1 1 1 1 1 0 1

ESS3 – select and nested select statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

select *name* [in *words* ...];
do *commands*; done
	select fname in *;
	do
	echo picked $fname \($REPLY\)
	break;
	done
		1
		0
		1
		1
		0

ESS4 – trap

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
declare -t VARIABLE=value trap "echo VARIABLE is being used here." DEBUG # rest of the script	trap "echo Booh!" SIGINT SIGTERM echo "pid is $$" while : # This is the same as "while true". do sleep 60 # This script is not doing anything. done	1 0 1 1 0 0 1 0 0

ITERATION Statements

EIS1 – for-do

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

for *name* [in *words* ...]; do *commands*; done
 	
for ((*expr1 ; expr2 ; expr3*)); do *commands*; done
	# Loop through a set of strings:
	for m in Apple Sony Panasonic
	"Hewlett Packard" Nokiado; do
	
	echo "Manufacturer is:" $m; done
	
	# or as a single line...
	for m in Apple Sony Panasonic
	"Hewlett Packard" Nokia;
	do
	echo "Manufacturer is:" $m; done
	
	# Loop 100 times:
	for i in $(seq 1 100);
	do
	echo -n "Hello World${i} "; done
	
	# Loop through the arguments passed
	to a function:
	foo (){
	   for ARG in "$@";
	do
	echo $ARG; done}
	# try it
		0
		1
		0
		
		1
		
		0
		1
		0
		0
		1
		
		0
		1
		0
		1
		
		0
		0
		1
		1
		0
		1
		0

EIS2 – while-do

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

while *test-commands*;
do *consequent-commands*;
done
	i="0"
	
	while [$i -lt 4]
	do
	   xterm &
	   i=$[$i+1]
	done
		1
		
		1
		0
		1
		1
		0

EIS3 – until-do

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

until *test-commands*;
do *consequent-commands*;
done
	myvar=0
	until [$myvar -eq 10]
	do
	   echo $myvar
	   myvar=$(($myvar + 1))
	done
		1
		1
		0
		1
		1
		0

JUMP Statement
(are counted as they invoke action – pass to the next statement)

EJS1 - return

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
return [*n*]	return $abc	1

EJS2 - break

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
break	echo "OK, see you!" break	1 1

EJS3 – exit function

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

exit	if test "$1" == "" ; then
	   echo $0 BAR
	   exit
	fi
		1
		1
		1
		0

EJS4 - continue

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

continue	if [["$name" != *[[:upper:]]*]];
	then
	continue
	fi
		1
		0
		1
		0

EXPRESSION Statement

EES1 – function call

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<function_name>	read_file	1

EES2 – assignment statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<name> = <value>;	"$1" =”one"	1

EES3 – empty statement (counted as it is considered to be a placeholder for something to call attention)

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
one or more “;” in succession	;	1 each

BLOCK Statement

EBS1 – block means related statements treated as a unit

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

# start of block 	
{
<definitions>
<statement>
}
# end of block 	
	# start of block
	{
	$i = 0;
	echo “hi”
	}
	# end of block
		0
		0
		1
		1
		0
		0

DECLARATION OR DATA LINES

DDL1 – variable declaration

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<keyword><option><variable>	declare -i number	1

4. Notes on Special Character Processing

1) Quotes:

Start of Quotes:	"\""
End of Quotes:	"\""
Escape Front Quotes:	'\\'

2) Line Continue: "\\"

3) Two types of file extensions are recognized for Bash: “.sh” and “.ksh”