Bash Shell Script CodeCount™

Counting Standard

University of Southern California

Center for Systems and Software Engineering

Spring 2010

Revision Sheet

Date	Version	Revision Description	Author
05/11/10	1.0	Original Release	CSSE

1.0 CHECKLIST FOR SOURCE STATEMENT COUNTS

PHYSICAL AND LOGICAL SLOC COUNTING RULES

Measurement Unit	Order of Precedence	Physical SLOC	Logical SLOC	Comments
Executable lines	1	One per line	See table below	Defined in 2.8
Non-executable lines				
Declaration (Data) lines	2	One per line	See table below	Defined in 2.4
Compiler directives	3	One per line	See table below	Defined in 2.5
Comments				Defined in 2.7
On their own lines	4	Not included (NI)	NI	
Embedded	5	NI	NI	
Banners	6	NI	NI	
Empty comments	7	NI	NI	
Blank lines	8	NI	NI	Defined in 2.6

Table 1 Physical and Logical SLOC Counting Counts

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 *{…}* and semicolon *;* used with this statement are not counted.
R03	Statements ending by a semicolon or newline	3	Count once per statement, including empty statement.	Semicolons within "*for*" statement 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 directive	5	Count once per directive.	

Table 2 Logical SLOC Counting Rules

2.0 DEFINITIONS

2.1 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.

2.2 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.

2.3 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.

2.4 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 Bash shell script keywords that denote data declaration lines:

Data Declaration
`declare`
`local`
`type`
`typeset`

Table 3 Data Declaration Types

2.5 Compiler directive – 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.

2.6 Blank line – A blank is a tab or space. What this actually means is - a blank is any chunk of white space between anything that is printable (a character or word). So a blank can be several spaces or tabs or a combination of multiples of the two.

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

Bash shell script comment delimiters are "**#**". A whole comment line may span one line and does not contain any compilable source code. An embedded comment can co-exist with compilable source code on the same physical line. Banners and empty comments are treated as types of comments.

2.8 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

  NOTE: See Section 3 of this document for examples of control 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

3.0 EXAMPLES OF LOGICAL SLOC COUNTING

EXECUTABLE LINES

SELECTION STATEMENTS

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT

ESS1	if, elif, else and nested if statements	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	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	select *name* [in *words* ...]; do
		commands; done
			select fname in *;
			do
			    echo picked $fname \($REPLY\)
			    break;
			done
				1
				0
				1
				1
				0

ESS4	trap	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

ITERATIONS STATEMENTS				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**

EIS1	for-do	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	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	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 STATEMENTS (ARE COUNTED AS THEY INVOKE ACTION – PASS TO THE NEXT STATEMENT)				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**
EJS1	return	return [*n*]	return $abc	1
EJS2	break	break	echo "OK, see you!" break	1 1
EJS3	exit function	exit	if test "$1" == "" ; then echo $0 BAR exit	1 1 1

DECLARATION (DATA) LINES				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**
DDL1	variable declaration	<keyword><option><variable>	declare -i number	1