ACL: Automated Command Line Language Reference Manual

Cheow Lip Goh (Paul)
UNI: clg2111
Email: cheowlipgoh@gmail.com

1. Introduction

ACL is created to make it very easy to do repetitive and tedious tasks on the command line. The syntax is very similar to the widely used C/Java, making it a very easy language to pick up by most people. The ACL interpreter is based on java and has a good potential to be used on different platforms.

2. Lexical Conventions

ACL has 5 kinds of tokens: identifiers, keywords, constants, expression operators, and other separators. Blanks, tabs, newlines and comments are ignored unless they serve to separate tokens.

2.1 Comments

The first style of comments starts with /* and terminates with */. The seconds style starts with // and terminates with end of line.

2.2 Identifiers

An identifier is a series of letters and digits. The first character of an identifier must be a letter, followed by any number of letters or digits or underscore. Upper and lower case letters are distinct.

2.3 Keywords

The following table lists all the keywords in ACL. Keywords are reserved identifiers and should not be used for any other purpose:

session	receive	send	if
else	while	for	function
return	break	continue	

2.4 Constants

There are only two kinds of constants in ACL, namely Integer and String.

2.4.1 Integer Constants

An integer constant is a sequence of digits.

2.4.2 String Constants

A string constant is a sequence of characters surrounded by double quotes “. The forward slash followed by a double quote \” is used to represent a double quote within the string.

2.5 Other Tokens

```
(       )       {       }       ;       ,
+       -       *       /       ++      --
>       <       >=      <=      ==      =
```

3.0 Expressions

3.1 Primary Expressions

Primary expressions include identifiers, constants, parenthesized expressions and function calls.

3.1.1 Identifiers

An identifier could be an lvalue expression or rvalue expression. An rvalue expression will be evaluated.

3.1.2 Constants

A constant's value determines its type (i.e., Integer or String). A constant can only be an rvalue.

3.1.3 Parenthesized Expressions

Parenthesized expressions' type and value are identical to the unparenthesized expressions.

3.1.4 Function Calls

Function calls consist of a function identifier followed by a parenthesized list of arguments to the function. Each argument is an expression. The function call itself is an rvalue expression.

3.2 Arithmetic Expressions

Arithmetic expressions take primary expressions as operands and evaluate those using operators.

3.2.1 Binary Operators

Binary arithmetic operators include +, -, *, /. They indicate addition, subtraction, multiplication and division. Binary operators can only operate on Integer constants operands. Multiplication and division operators have a higher precedence than addition and subtraction operators. When the precedence of the operators are the same, i.e., addition and subtraction, the associatively is from left to right.

3.3 Relational Expressions

Relational operators >, <, >=, <=, !=, == are binary relational operator representing whether the first operand is greater than, lesser than, greater than or equal, lesser than or equal, not equal and equal to the second operand. These operators evaluate to 0 if true and 1 if false.

3.4 Assignment Expressions

The assignment operator is =. The assignment expression consist of an lvalue to the left of the assignment operator, and an rvalue to the right of the assignment operator. The lvalue must be a modifiable identifier. The rvalue can be any identifier or constant.

4.0 Statements

Except as indicated, statements execute in sequence.

4.1 Expression Statement

Most statements are expression statements, which have the form:

expression ;

Expressions statements are usually assignments or function calls.

4.2 Compound Statement

Compound statement is a group of statements surrounded by open and close braces. It is used when a group of statements is needed in place of a single statement.

4.3 Conditional Statement

The two forms of condition statements are:

If (expression) then statement
If (expression) then statement else statement

In the first form, the first substatement is executed if the expression evaluates to a non-zero. In the second form, the second substatement is executed if the expression evaluates to zero. The else ambiguity is resolved by connecting the else to the last encountered elseless if.

4.4 While Statement

The while statement has the form:

while (expression) statement

As long as the value of the expression remains non-zero, the substatement is executed repeatedly

4.5 For Statement

The for statement has the form:

for (expression-1 ; expression-2 ; expression-3) statement

expression-1 specifies the initialization for the loop.
expression-2 specifies the test to perform such that as long as the expression-2 evaluations to non-zero, the for loop will continue to run.
expression-3 specifies the incrementation made after the end of each loop iteration.

4.6 Break Statement

The break statement has the following form:

break ;

If executed, the break statement will cause the termination of the smallest enclosing while or for statement. Execution control will pass on to the statement following the terminated statement.

4.7 Continue statement

The continue statement has the following form:

continue ;

If executed, the continue statement will pass control to the loop continuation portion of the smallest enclosing while or for statement.

4.8 Return statement

The return statement two forms:

return ;
return expression ;

The return statement is always used within a function. The first form of return statement returns no value. The second form of the return statement will return the value of the expression to the function caller.

5.0 Function Definition

A function definition has the following form:

```
function function-name ( argument-list ) {
        statement
}
```

function-name is the name of the function. argument-list is a comma separated list of arguments to be passed into the function. argument-list could be empty. The return statement is optional and could be used when the function needs to return, with or without a value.

6.0 Built-in Functions

ACL has 3 built in functions, namely session, receive and send. These 3 functions are the key to automatically interacting with the command line locally or remotely.

6.1 session function

The session function, when invoked, will spawn a local or remote command line session, depending on the shell program used to spawn the session (i.e., telnet, ssh, etc). The session function is invoked just like a regular function and takes an identifier or a string constant. If an identifier is used, the identifier must evaluate to a string.

6.2 receive function

The receive function's invocation is only meaningful after the session function is called. The receive function will wait for a certain sequence of characters to be sent back by the spawned session to determine if there is a match. The receive function is invoked just like a regular function and takes an identifier or a string constant. If an identifier is used, the identifier must evaluate to a string. The default timeout for the receive function's wait is 30 seconds. This value could be changed by the timeout function described later. If a timeout happens during receive, a timeout error will be printed on the screen.

6.3 send function

The send function’s invocation is only meaningful after the session function is called. The send function will send a certain sequence of character to the spawned session. The send function is invoked just like a regular function and takes an identifier or a string constant. If an identifier is used, the identifier must evaluate to a string.

6.4 timeout function

The timeout function is used to set the timeout for the receive function. The timeout function is invoked just like a regular function and takes an identifier or an integer constant. If an identifier is used, the identifier must evaluate to an integer.