Brash Reference Manual

Robert Clarke

November 23, 2022

Contents

1 A Foreword

1.1 Motivations

1.2 What Brash is

1.3 What Brash isn't

2 Introduction to Brash Syntax

2.1 Basic Types

There are three basic types in Brash: **Number**, **Boolean**, and **String**.

- Number values are base-10 double precision floating point numbers (Supports up to 6 decimal places)
- Boolean values are either **true** or **false**
- String values are series of textual characters that are enclosed by double-quotation marks

Listing 1: Basic Types

```
//Numbers
4
25
-2
33.333

//Booleans
true
false

//Strings
"This is a string!"
"Good morning"
```

2.2 Variables

To assign a name to a value, use variables for easy reference in your program. To declare a variable use the **var** keyword, followed by an identifier, an equals sign, and the value you wish to bind to the variable. After a variable is declared, you may reference the variable's value without restating the **var** keyword.

Listing 2: Variables

```
var name = "Greg"
var age = 20
age = age + 1
var canDrive = true

print name    //Displays 'Greg'
print age     //Displays '21.000000'
print canDrive //Displays 'true'
```

2.3 Expressions and Operators

An expression is a series of values and operators which evaluate to a single value. A majority of the operators in Brash are 'infix' operators which appear between values in the form "A OPERATOR B". However, there are some prefix operators which appear before the value they modify.

- Special operators
  - Assignment = The assignment operator binds the variable identifier of its left operand to the value of its right operand
  - Grouping () The grouping operators signal that the enclosed expression should be evaluated first
- Arithmetic operators
  - Addition +
  - Subtraction -
  - Multiplication *
  - Division /
  - Modulo %
  - Negation - (prefix)
- Comparative operators
  - Lesser <
  - Lesser-Equals <=
  - Greater >
  - Greater-Equals >=
- Identity operators
  - Equality ==

  - Inequality !=

- Logical operators
  - Logical NOT ! (prefix)
  - Logical AND &&
  - Logical OR ||
  - Logical XOR ^^

2.3.1 Operator Precedence

The order in which operators are evaluated are determined by their precedence. Higher precedence operators are evaluated before lower precedence operators.

Table 1: Precedence - High to Low

Operator	Operator Name	Associativity
()	Group	None
!	Logical NOT	Right
-	Arithmetic Negation	Right
*	Multiplication	Left
/	Division	Left
%	Modulo	Left
+	Addition	Left
-	Subtraction	Left
&&	Logical AND	Left
||	Logical OR	Left
^^	Logical XOR	Left
<	Lesser	None
>	Greater	None
<=	Lesser-equal	None
>=	Greater-equal	None
==	Equality	None
!=	Inequality	None
=	Assignment	Right

2.4 Control Flow

While you can make useful programs with the structures discussed so far, the magic really begins when your programs can take different actions based on certain conditions.

The **if** keyword must be followed by an expression that evaluates to a boolean value. When the conditional expression is true, the statements within the if-block are executed. When the conditional expression is false, the if-block is skipped.

Listing 3: If Statement

```
var waffles = 7
if waffles > 3 {
    print "Thats's a lot of waffles!"
}
print "Enjoy your breakfast!"
```

If you have prior experience in the C family of programming languages, you may prefer to enclose your conditions with brackets:

Listing 4: If Statement (C-style)

```
var waffles = 7
if (waffles > 3) {
    print "Thats's a lot of waffles!"
}
print "Enjoy your breakfast!"
```

To execute instructions when an **if**'s conditional expression is false, follow the **if** block with an **else** block.

Listing 5: If-Else

```
var waffles = 2
if (waffles > 3) {
    print "Thats's a lot of waffles!"
}
else {
    print "That's a sensible amount of waffles!"
}
print "Enjoy your breakfast!"
```

2.5 Loops

The **while** keyword is followed by an expression that evaluates to a boolean value. The loop body follows after the conditional expression. Once all the statements within the loop body have been executed, the conditional expression is re-evaluated. If the conditional expression is false, skip the loop body and resume execution at the next statement.

Listing 6: While Loop

```
var count = 0
while count < 10 {
    print count
    count = count + 1
}
print "All done!"
```

3 Bytecode Details

Brash code gets compiled to bytecode before interpretation. If stored as a standalone file it is called a Brash Object File, and **.bro** is the preferred file extension.

3.1 Numbers

When encoding numbers into bytecode, they are encoded as 8-byte double-precision floating point numbers. Byte order is Big-Endian a.k.a. TCP/IP Network Order.

3.2 Strings

When encoding strings into bytecode, they are encoded as null-terminated strings.

3.3 Functions

When encoding a function definition into bytecode, functions are encoded as follows:

- One-byte Function Definition Opcode
- Byte-length of function definition and function body: one 16-bit unsigned integer
- Encoded String of the function's name (See 3.2)
- Arity: one 8-bit unsigned integer
- If function arity is 0 the following sub-items are omitted, otherwise each sub-item repeats a number of times equal to arity
  - Encoded Type of n-th parameter
  - Encoded String of the n-th parameter's name (See 3.2)
- R-Arity (Return Arity): one 8-bit unsigned integer
- If function r-arity is 0 the following sub-item is omitted, otherwise the sub-item repeats a number of times equal to r-arity
  - Encoded Type of n-th returned item
- Function body: continue compilation as normal