

# RPGsh User Manual

TheMohawkNinja

March 29, 2024

Ver. 0.8.0

**DISCLAIMER:** This project is entirely developed independantly. I am not associated in anyway with Wizards of the Coast, Paizo, or any other game development company. To any lawyers eyeing me up, I will not be adding information that would negate the need for players to purchase your products (e.g. game rules).

# Contents

<b>1</b>	<b>Introduction and Basic Usage</b>	<b>3</b>
1.1	The Prompt . . . . .	3
<b>2</b>	<b>Program Listing</b>	<b>4</b>
<b>3</b>	<b>Variables</b>	<b>5</b>
3.1	scope . . . . .	5
3.2	type . . . . .	6
3.2.1	var . . . . .	6
3.2.2	dice . . . . .	8

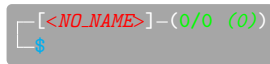
# 1 Introduction and Basic Usage

The Roleplaying Games Shell, `rpgsh`, is an interactive and extensible shell purpose-built for augmenting player and DM gameplay for table-top RPGs like Dungeons & Dragons<sup>®</sup>, Pathfinder<sup>®</sup>, and more!

`rpgsh` provides users with capabilities similar to those found in conventional shells like `bash` or `PowerShell` like command execution and variable assignment/modification, while also adding functionality more unique to shell environments like varying data types and variable scopes.

## 1.1 The Prompt

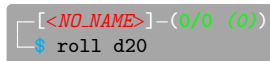
When interacting with the shell directly, you will be presented with a prompt that will look similar to the following:



```
[<NO_NAME>]-(0/0 (0))
$
```

The prompt contains the currently loaded character's name (`<NO_NAME>`) along with their **current/max** (*temp*) hitpoints.

As with any command line interface, you interact with the prompt by entering in either a variable or a program, along with any operators or parameters. For example, if you want to roll a 20-sided die, you would enter the following:



```
[<NO_NAME>]-(0/0 (0))
$ roll d20
```

The maximum size of the input buffer for the prompt is 256 characters. Exceeding this will throw an error.

## 2 Program Listing

As of version 0.8.0, the following programs are available to the user when interacting with the `rpgsh` prompt:

### `banner`

Displays the ASCII art logo for `rpgsh` along with a one-line description of the program and the author's signature.

### `help`

Lists all applications available to `rpgsh` along with a brief description.

### `list` \*

Lists all the variables in one or all scopes.

### `roll` \*

Dice-rolling program which supports custom lists and result counting.

### `setname` \*

Sets which variable is used for displaying the character's name.

### `variables`

This is ***NOT*** to be explicitly called by the user, but is instead implicitly called when the user enters a variable as the first parameter in the prompt.

### `version`

Prints `rpgsh` version.

\*Program contain additional parameters. Additional information can be found by running the program with the `-?` or `--help` flag.

## 3 Variables

`rpgsh` allows the user to set, get, and modify variables. Variables are arranged in a nested hierarchy through three different scopes, and through an arbitrary number of levels within each scope. Operations may be performed on variables, with the following operators currently supported:

Binary: `=`, `+`, `-`, `*`, `/`, `+=`, `--`, `*=`, `/=`

Unary: `++`, `--`

The components of an operation must be space delimited as shown below:

`$v/four = 2 + 2` **Correct**

`$v/four=2+2` **Incorrect**

Variables in `rpgsh` follow the below syntax:

`<scope><type>[<character>]/<level 1>/<level 2>/.../<level n>`

Below describes each part in detail:

### 3.1 scope

A single character representing which level of the overall hierarchy is being referenced. There are three total scopes for use in `rpgsh`. These are defined below:

- @** Character attributes. This scope encompasses all variables specific to a given character. If the `<character>` attribute is omitted, this references the currently loaded character.

These are stored in `~/.rpgsh/campaigns/<campaign>/characters/<charactername>.char`

- #** Campaign variables. This scope encompasses all variables in the current campaign, and are available while any character in the current campaign is loaded.

These are stored in `~/.rpgsh/campaigns/<campaign>/.vars`

- \$** Shell variables. This scope encompasses all campaigns and is the broadest scope in `rpgsh`.

These are stored in `~/.rpgsh/.vars`

## 3.2 type

A single character representing the data type of the variable. When calling an existing variable, this parameter may be omitted, in which case `rpgsh` will find the matching variable. If more than one match is found, the first match will be used. As of version 0.8.0, the following data types have been implemented: `v` (var), `d` (dice), `c` (currency), `s` (currency system), and `w` (wallet). Note that in all operations, the data type of the returning value will always be the same as the left-hand side (LHS) of the operation. The data types are defined below:

### 3.2.1 var

These are generic, lazily-evaluated variables that may contain either a string or an integer, similar to how variables in many scripting languages operate. Operations performed on var-type data objects are thus dependant on whether or not the current value stored is evaluated to be a string or an integer.

For example, if you want to initialize a var character attribute called "MyVar" set to the number three, you would enter:

```
[<NO_NAME>]-(0/0 (0))
$ @v/MyVar = 3
```

Alternatively, if you want to initialize that same variable to the string "three", you would enter:

```
[<NO_NAME>]-(0/0 (0))
$ @v/MyVar = three
```

If you want assign a string containing spaces to a variable, it must be wrapped in quotation marks, for example:

```
[<NO_NAME>]-(0/0 (0))
$ @v/MyVar = "The number three"
```

Below are the operations tables for var-type variables, the top table for var-type variables which evaluate to an integer, and the bottom for var-type variables which evaluate to a string. Each table describes the result of an operation when the right-hand side (RHS) is of each data type. Cells marked with an **ERR** result in an error being thrown and no change being made to the LHS.

LHS evaluates to Integer						
<i>Op</i> ( <i>Bin.</i> )	<b>v</b> ( <i>Integer</i> )	<b>v</b> ( <i>String</i> )	<b>d</b>	<b>c</b>	<b>s</b>	<b>w</b>
<b>=</b>	Assignment	Assignment*	<b>ERR</b>	<b>ERR</b>	<b>ERR</b>	<b>ERR</b>
<b>+</b>	Addition					
<b>-</b>	Subtraction					
<b>*</b>	Multiplication					
<b>/</b>	Division					
<b>+=</b>	Addition Assignment					
<b>-=</b>	Subtraction Assignment					
<b>*=</b>	Multiplication Assignment					
<b>/=</b>	Division Assignment					
<i>Op</i> ( <i>Un.</i> )	—					
<b>++</b>	Increment					
<b>--</b>	Decrement					

LHS evaluates to String						
<i>Op</i> ( <i>Bin.</i> )	<b>v</b> ( <i>Integer</i> )	<b>v</b> ( <i>String</i> )	<b>d</b>	<b>c</b>	<b>s</b>	<b>w</b>
<b>=</b>	Assignment*	Assignment	ERR	ERR	ERR	ERR
<b>+</b>	ERR	Concatenation				
<b>-</b>		ERR				
<b>*</b>						
<b>/</b>		Concatenation				
<b>+=</b>						
<b>-=</b>		ERR				
<b>*=</b>						
<b>/=</b>						
<i>Op</i> ( <i>Un.</i> )		—				
<b>++</b>	ERR					
<b>--</b>	ERR					

\*A warning will be thrown to indicate that the evaluated data type has changed.

### 3.2.2 dice

These are variables which contain a string of characters in the standard RPG dice format of  $CdF[+|-]M$ , where:

$C$ : (optional, assumes a value of 1 if omitted) The count (quantity) of dice

$F$ : The number of faces of the di(c)e

$M$ : (optional) A modifier value

For example, if you want to initialize a dice character attribute called "MyDice" representing two six-sided dice with a modifier of +3, you would enter:

```
[<NO_NAME>]-(0/0 (0))
$ @d/MyDice = 2d6+3
```

The main purpose of having a distinct data type for what could otherwise be accomplished via a var-type variable is due to the different operations that can be performed to change the properties of the dice as is described by the below operations table:

$Op$ (Bin.)	<b>v</b> (Integer)	<b>v</b> (String)	<b>d</b>	<b>c</b>	<b>s</b>	<b>w</b>
<b>=</b>	<b>ERR</b>	Assignment**	Assignment	<b>ERR</b>	<b>ERR</b>	<b>ERR</b>
<b>+</b>	Modifier += RHS	<b>ERR</b>	Count += RHS*			
<b>-</b>	Modifier -= RHS		Count -= RHS*			
<b>*</b>	Count *= RHS		<b>ERR</b>			
<b>/</b>	Count /= RHS		<b>ERR</b>			
<b>+=</b>	Modifier += RHS		Count += RHS*			
<b>-=</b>	Modifier -= RHS		Count -= RHS*			
<b>*=</b>	Count *= RHS		<b>ERR</b>			
<b>/=</b>	Count /= RHS					
$Op$ (Un.)	—					
<b>++</b>	Modifier++					
<b>--</b>	Modifier--					

\*If and only if both dice have equal faces. Additionally, in the event that both dice have different modifiers, a warning will be thrown indicating that only the LHS modifier will be preserved.

\*\*If and only if the string is formatted appropriately, otherwise an error will be thrown.