

# QDL Scripting

Version 1.3.1

## Introduction

Scripting – the execution of a set of commands that do not require compilation – is included in QDL. It has many features to help with this. The basics are that any set of QDL commands in a file (extension is .qdl) can be run using your current workspace settings, just as if you were typing it in.

## The basics

The requirements are

- install QDL. It is assumed installed in \$QDL\_HOME.
- set your preferred configuration in the \$QDL\_HOME/bin/qdl-run script.
- Probably add \$QDL\_HOME/bin to your \$PATH so you can just type qdl or qdl-run as needed.

Once you have these, a good test is to execute your first script:

```
qdl-run $QDL_HOME/examples/hello_world.qdl  
Hello World!
```

Which runs the basic hello world script.

If you want to play with the examples more easily, you should probably set *script\_path* in the configuration (see the document in the docs directory or look at the online version). Then all you need to do is enter the name of the script to run it within the workspace. Note that you will need to pass in the correct file name to qdl-run since the operating system handles that and is unaware of things like virtual file system in QDL.

## Running vs. loading scripts

We use two similar terms, *run* and *load*. The difference are

- *run* implies that any state/variables created in the script remain in the script. The current environment is made available (so VFS's, modules, variables, user functions.) Anything the script creates goes away when it ends.
- *load* executes the contents of the script in the current environment. This is useful for, e.g., initializing a bunch of variables, or loading things into the current environment. At the end of the script, everything it did is now available. Initialization scripts and boot scripts are generally loaded.

Mostly you want to run scripts rather than load them. It is always better to keep your environments separate unless you know exactly what the script does. An unknown script may reset or overwrite your current environment with no warning if loaded.

## From the command line: passing and using arguments

You may pass arguments to the script as

```
qdl-run script_name arg0 arg1 arg2,...
```

Note: It is often a good idea to enclose arguments in double quotes and in point of fact, if there are embedded blanks it is mandatory. These may then be accessed inside the script using the `script_args()` function. Note that these are all strings since that is the only option supported. (If you object, please lobby to have your OS rewritten to be QDL aware.) the `script_args()` call is documented more fully in the documentation (included in `$QDL_HOME/docs/qdl_reference.pdf`) but the basics are

- `script_args()` – no arguments returns how many there are, e.g. 3
- `script_args(n)` – `n` is an integer in the range  $0 < n < \text{script\_args}()$  will return the `n`-th argument.

### Example

```
qdl-run script_name fee fi fo fum
```

would have the following

`script_args()` returns 4

`script_args(0)` return the string 'fee'

`script_args(3)` returns 'fum'

## Calling scripts in QDL: passing and using arguments

To run a script from another script, use the `script_run()` call:

```
script_run('path/to/script.qdl' [,arg_list.]);
```

So the script at the given path would be invoked. The list of arguments is a stem list. These may be anything – unlike invocation from a shell script, where you may only pass strings. A very useful trick is to use the `to_list()` call on the arguments you want to pass:

```
script_run('script_name', to_list('fee','fi','fo','fum'))
```

## Argument lifecycle

Once arguments are given, they are available to that script and any scripts it calls. If a sub-script has its own set of arguments, it has those but the parent's arguments are not changed. In this way arguments may be inherited.

## Getting input from users

QDL has a function `scan`:

```
scan([prompt])
```

This will print *prompt* if present and return a string of whatever the user types in. E.g.

```
say(scan('type something>'));  
type something>mairzy doates and dozey doats.  
mairzy doates and dozey doats.
```

So if your script requires some user input, you may use this. There is a script in the `$QDL_HOME/examples` directory named `scan_it.qdl` with a good example.

## Returning values

Scripts may return values using the `return()` call. There are a few items to note.

## External facing scripts

If this is to be a script that is outward facing (so consumed by another process), any result returned will be sent to standard out plus a line feed (so it will display properly at the console). Standard out best understands strings, so we use them. While you can return your massive stem, it will get converted to a string. This is a limitation of underlying OS's. In this case, it is not a bad idea at all to convert it to JSON since that is one reason JSON exists. Not having a return statement or calling `return()` (no argument) does not return a result to standard out.

```
bash$ export QX=$QDL_HOME/examples  
bash$ qdl-run $QX/sqrt.qdl 5  
4.99999999999999968  
bash$
```

## Inward facing scripts

If the script is called by other scripts, you can return anything and that will be the output value of the script, *e.g.* if you have a script that computes approximations of a square root for an argument (this is included in the examples directory – the *script\_path* in the configuration has been set to look there):

```
script_run('sqrt.qdl', to_list(5))^2  
4.99999999999999968
```

## Calling other scripts

Once a script is running in QDL, it has full access to the resources of the workspace. So that means that it can call other scripts that are, *e.g.*, in a VFS. A common pattern is to have a boot script that simply starts QDL and loads other scripts to do the work.

## Example

You have a script that resides in a VFS at `vfs#/bsu/scripts/init.qdl`. You need to pass it several arguments that only are external. Since all arguments are passed down to any subsequent scripts, they are available. How to run it from the command line? Write a script like `bsu-init.qdl`:

```
script_run('vfs#/bsu/scripts/init.qdl');
```

(Yes, that is the entire script, a single line.) Note that there are no arguments – if you supplied them, they would over-write what was passed in. When you invoke this through `qdl-run`, pass in what you want, e.g.,

```
qdl-run bsu-init.qdl  
/grid/certs/03456564.pem  
https://shibboleth.bigstate.edu/secure  
"openid profile email"
```

(line breaks added for readability.) and these three arguments are available to the script via `script_args()`.

Scripts may call other scripts and there is no limit placed. Much like recursive functions, they may call themselves and this is not flagged as an error. However, recursion is generally a terrible idea if it happens without being an explicit design decision.