Common Framework For Unix Scripting Languages

Sumit Dubey
Department of Computer Science
Manipal University
256 (Ground Floor), Okhla Industrial Estate-III,
New Delhi - 110020, India
sumitgpl@gmail.com

Kanchan Lata
Faculty of CS/IT
Dewan V.S.Institue of Engineering &Technology
Partapur Bypass Road, Meerut (U.P.)
kanchan.jssit@gmail.com

***Abstract*—With the thousands of commands available for the command line user to write own application based on some complex shell script or other script. The complexity implies more difficulties to make an efficient monitoring, managed and especially fault diagnosis system. We propose a new framework for the programmer/ user to create a well managed system with having capabilities of debugging and easy to understand for other person. The interest of this framework is to combine all type of UNIX script programming language into one system.**

Keywords- Framework, Scripting languages

I. Introduction

A shell script is a script written for the shell, or command line interpreter, of an operating system. The shell is often considered a simple domain-specific programming language. Typical operations performed by shell scripts include file manipulation, program execution, and printing text.

Many shell script interpreters double as command line interface, such as the various UNIX shells, Windows Power Shell or the MS-DOS COMMAND.COM. Others, such as AppleScript or the graphical Windows Script Host (WScript.exe), add scripting capability to computing environments without requiring a command line interface. Other examples of programming languages primarily intended for shell scripting include DCL and JCL.

Like the Shell scripts all Scripting languages allow us to program commands in chains and have the system execute them as a scripted event, just like batch files. They also allow for far more useful functions, such as command substitution. We can invoke a command, like date, and use its output as part of a file-naming scheme. We can automate backups and each copied file can have the current date appended to the end of its name. Scripts aren't just invocations of commands, either. They're programs in their own right. Scripting allows us to use programming functions – such as 'for' loops, if/then/else statements, and so forth – directly within your operating systems interface. And, we don't have to learn another language because you're using what we already know: the command-line.

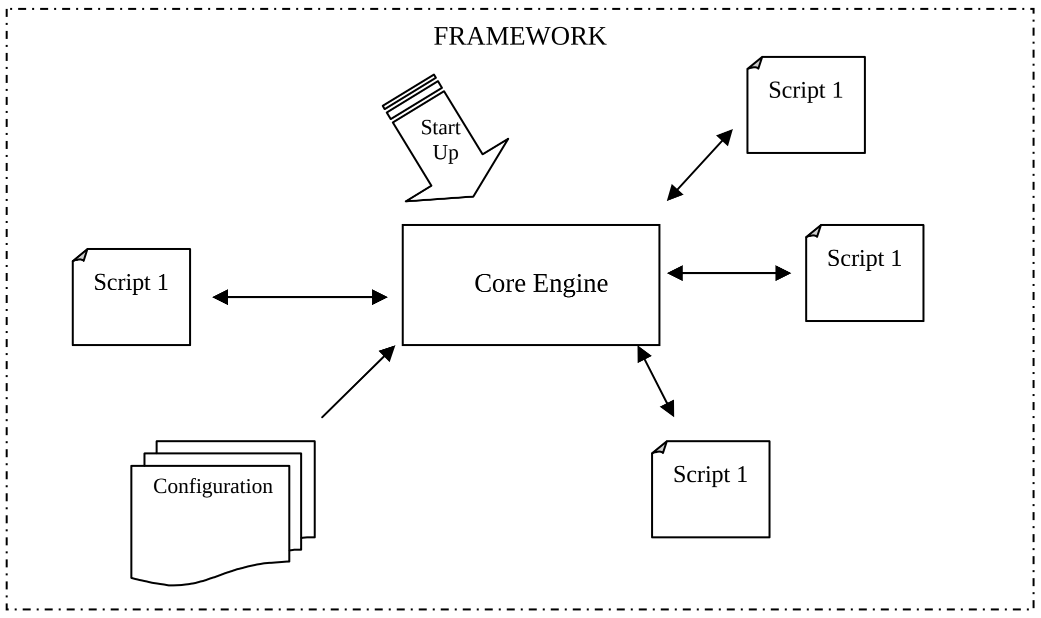
But writing programs into shell script is not sufficient. Some time we required having more complex process and wanting to do these into very simple way, and then we required low level language like C, C++, Perl, Python etc. when we have multiple flavors of programs then its very complex to handle and hard to understand the flow of process. For such type of complex process we required one fixed flow and easy to implement framework.

When we are talking about framework then definitely below some points should come into mind….

- Ability to create scriptable applications.
- Ability to user can write own scripts and own standard.
- Ability to communicate between other scripts.
- In advance level programming, programmer able to make some changes into core engine as well as able to add some new standard into core engine.
- User can define own program type (like Perl, Python, bash shell, K shell etc).
- User can define own flow of process.

II. System Structure

A. *High Level Structure*

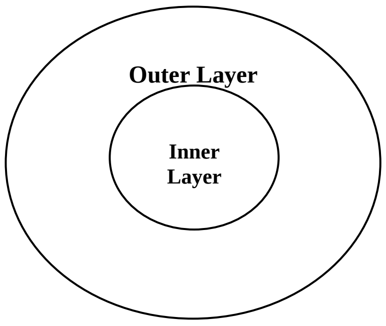


High Level Structure

As per the above questions, if we are going to draw a system structure that should be look like above figure. In the above high level structure scripts showing any type of scripting / programming language like Perl, Python, Shell script or c/c++ etc. These script directly interacting with core engine and core engine has ability to execute them based on defined configurations. This configuration may have header information about every script as well their relationship between them.

To start application we required some entry point, from where application can start and after that can call other scripts also. Same way in this high level structure we required some entry point, that showing with start up symbol. That entry point we can define into configuration file.

B. *Parts of Framework*



Parts of Framework

Our system we can define at two layers. One is outer layer and second is inner layer.

- Outer Layer (User scriptable area)

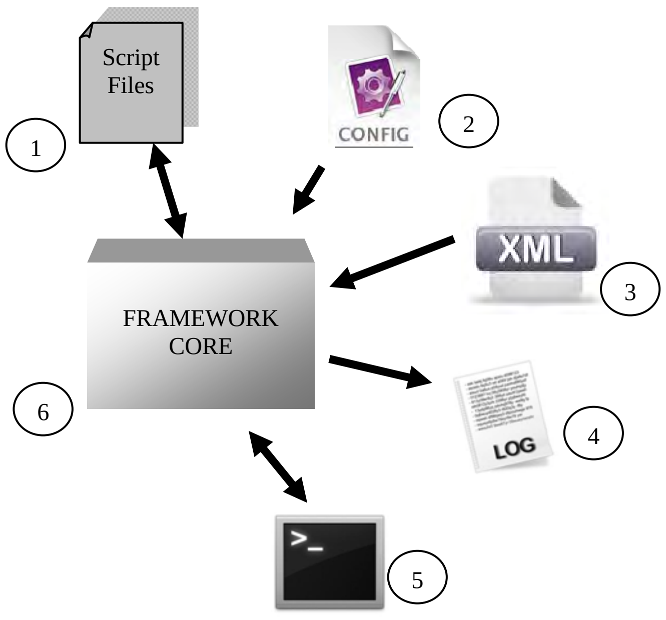
This layer is for the user scriptable area. As we see into high level structure, we have multiple scripts and configuration file to interact between them. That means user is free to add multiple scripts and their configuration as per their requirement and the core engine will execute them based on configuration.

- Inner layer (Framework core engine)

In the high level structure we have one core engine; this core engine is nothing but having ability to execute all types of scripts one by one as per defined sequence. That means this core engine is a group of program that doing all above activity. Group of program, means that should be based on some programming language, having some predefined methods and some sequence of execution of them. This group of program or core engine is the Inner layer and this is the core part of Framework.

Inner layer having some predefined group of program and we can define more based on our requirement. We will see more details into structure of inner layer part.

1) Structure of Outer Layer



Structure of Outer Layer

a) Scripts files

This scripts file can be shell script, Perl, Python or c / c++ file.

b) Config file

This configuration file will be fixed for an application and should be name as application name. Here we can decide the common global variable for the whole project/ application. Here we can also define the some global parameter that will be applicable for the whole application.

c) XML file

This XML file showing, how many script files are registered for the application and what are their running sequences. During registration of any script file we need to decide the script type and running sequence. That sequence showing successor and predecessor scripts for every script. Here we can also define the entry point for the application.

Some scripts required parameter also, for that type of script we can define parameter variable as well value into this XML file. With the parameter details we can also give here some help information which will be visible during running job on terminal.

d) LOG file

This log files showing the application's log. We not required writing logic inside the every script to create a common log for the application. This logging functionality of this framework automatically creates a log with predefined name and pattern on the predefined location. This predefined name, pattern and location also defined into the Config file of the application.

e) Terminal/screen output

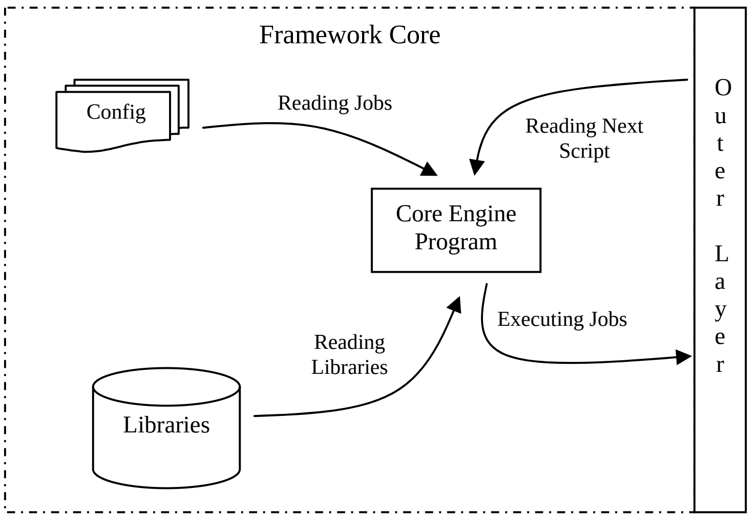
On running of every script, framework directing their output to the screen or terminal. This framework has capabilities to show the errors, warnings and other info of the running jobs on screen with different patterns, like for the error showing [ERROR] and for warning showing [WARN].

f) Framework Core

Having all the above required script and configuration files, framework should have some own initial methods and jobs to correlate all the scripts and run the whole application. Like the operating system kernel threads fork (), wait () etc we have jobs for handling all user defined script into framework core engine. In this framework there are some predefined jobs.

- init() – loading init Config.
- option()- checking XML file data and loading
- debug action()- set debug mode required environment
- help action ()- show help options
- exec action()- executing actions one by one
- clean() – cleaning the temporary storage
- validate() – check the basic syntax in framework
-

2) Structure of Inner Layer



Structure of Inner Layer

Above outer layer structure we got some details about Framework Core. This core is the inner layer of the framework. Init (), option (), validate () etc all the method comes under this inner layer and these method are written under some programming language and this programming language should be fixed. But no of method is not fixed, we can create more new method based on our requirement and able to plug into core engine. This inner core engine of framework having below units…

a) - Config

This Config showing framework core engine configuration. Here we have some predefined core engine configuration and its method using it during execution. As we know, we can add more method into core engine when we require, same as we can add more configuration tag into Config part.

In this Config part we have below things …

-- Here we can define some global variable like outer layer and read its value during all process.

-- Here we can define all methods with their running sequence and core engine will execute it.

-- Here we can define reference libraries for every method.

-- Here we can define interaction behavior between inner layer and outer layer.

b) - Core engine Program

This the main part of the framework and no one authorize to change this core part because this is the only program which is executing framework' core methods and based on these core method user can execute their defined scripts and jobs.

This core engine program's job to read all configurations for every method from Config and after linking with libraries, do the execution. That means there core engine program job is more critical because this is the only part where we are interacting outer layer with inner layer.

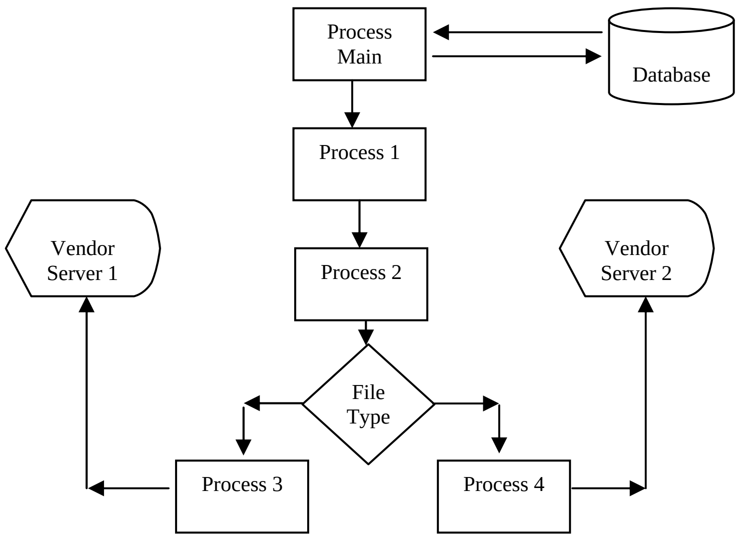
c) - Libraries

In this part we have stored libraries. This libraries may be binary form or may be the script form, its up to definition way. If we defining one new method into core engine, we required to put all referenced libraries into this Libraries part. Core engine program have capability to link method with their required library.

III. EXAMPLE

"Business wants a system, in which system should create predefined format file at the vendor server and vendor will process those file to do the goods fulfillment. These file should be encrypted and based on goods type file format and fulfillment processes will be decided."

Process would be like…



Process Main	
Type	Perl script
Task	Run database job and populate temporary tables

Process 1	
Type	Shell script
Task	• Create file from temporary table • File encryption • Rename file • Upload FTP server

Process 2	
Type	Perl script / Shell script
Task	• Decrypt the file • Read file type • Initiate process 3 or process 4 based on file type.

Process 3	
Type	Python Script
Task	Upload file to server 1 and restart some service.

Process 4	
Type	Python Script
Task	Upload file to server 2 and restart some service.

In the above business requirement, we can see there are 4 major processes those working conditions are different for the same target.

In the above system, process main reading database values from database and putting into some temporary table/ file and later on process 1 using same temporary table / file and creating a fulfillment file. This fulfillment file should be secured that's why here process 1 encrypting it, renaming it and uploading to FTP server. When we have encrypted file at FTP server then there is process 2 jobs to decrypt this file and based on file type (goods type) and initiate process3 or process 4. This process 2 may be on separate server or on the same server based on requirement because its take file input from FTP server not from direct process 1. That’s means we can schedule process 2 based on some time difference or we can direct trigger it from process 1 when process 1 completed its file creation on FTP. Process 3 and process 4 both are Python scripts and uploading this file to the vendor web server. After uploading process 3 or process 4 also doing some other web server activity.

Now let’s see this system into our framework

A. Scripts files

Process main (Perl), Process 1 (Shell script), Process 2 (Perl script), Process 3 (Python script), Process 4 (Python script), some c/c++ script for the database reading and file encryption/decryption.

B. Config file

Configuration file name would be <<VendorFulfillment.cfg>> and entry would be look like

```
#application name
APPNAME = VendorFulfillment
#scripting type
SCRPTTYPE = composite
#XML file location
XMLPATH = /export/home/sumit/vendorFulfillment/xml
#LOG file location
LOGPATH = /export/home/sumit/vendorFulfillment/log
#LOG file naming convention
LOGFILE= $appname$_$ddmmyyhhmiss$.log
#global variable with initial value
```

C. XML file

Some scripts required parameter also, for that type of script we can define parameter variable as well value into this XML file. With the parameter details we can also give here some help information which will be visible during running job on terminal.

```
<App>
        <Entry>Process Main</Entry>
        <Scripts>
                <Script name=”Process Main” Seq=1>
                <Type>Perl</Type>
                <Parameters>
                        <Parameter name=”databasenm” type=req val=”mydb”>
                        <Parameter name=”usernm” type=req>
                        <Parameter name=”passwd” type=req>
                <Parameters>
                <Help>
                        Process main help statement.
                </Help>
                </Script>
                <Script name=”Process 1” Seq=2>
                <Type>sh</Type>
                <Parameters>
                        <Parameter name=”tablenm” type=req>
                        <Parameter name=”norows” type=opt>
                <Parameters>
                <Help>
                        Process 1 help statement.
                </Help>
                </Script>
                <Script name=”Process 2” Seq=3>
                <Type>Perl</Type>
                <Parameters>
                        <Parameter name=”servernm” type=req>
                <Parameters>
                <Help>
                          Process 2 help statement.
                </Help>

                </Script>
                <Script name="Process 3" Seq=4>
                <Type>Python</Type>
                <Parameters>
                          <Parameter name="filenm" type=req>
                <Parameters>
                <Help>
                          Process 3 help statement.
                </Help>
                </Script>
                <Script name="Process 4" Seq=5>
                <Type>Python</Type>
                <Parameters>
                          <Parameter name=" filenm" type=req>
                <Parameters>
                <Help>
                          Process 4 help statement.
                </Help>
                </Script>
        </Scripts>
</App>
```

D. LOG file

As we already defined the log file location into Config file, Framework automatically will create a log file with defined file format $appname$_$ddmmyyhhmiss$.log. This file format showing combination of $appname$ and $ddmmyyhhmiss$ framework's predefined variables. $appname$ showing name of application and $ddmmyyhhmiss$ showing timestamp format in day, month, year, hour, minute and second.

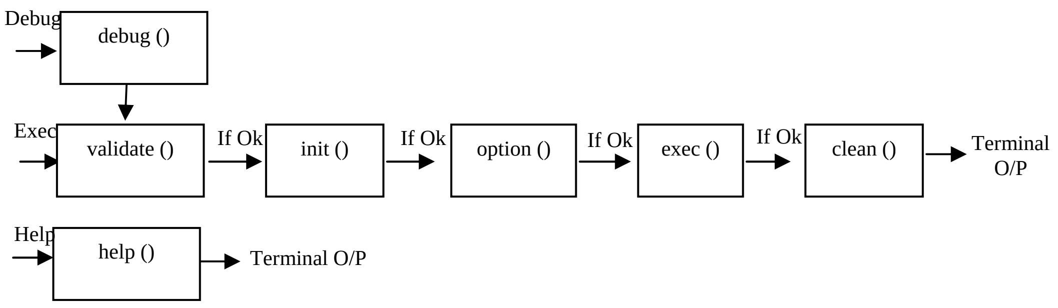
E. Terminal/screen output

On running of every script, framework directing their output to the screen or terminal. Like if process 1 doesn't have rights to write file on FTP server, then on terminal there should be an error message populated with exact error definition. This framework has capabilities to show the errors, warnings and other info of the running jobs on screen with different patterns, like for the error showing [ERROR] and for warning showing [WARN].

F. Framework Core

Now Framework core's job to execute all the process one by one based on defined sequence. That means this core will read first Config file and load the entire required configuration into buffer. Now time to read the XML file and based on sequence executing one by one process.

So Framework core's predefined methods will execute on below sequence …



IV. BENEFITS

- Reusable functions / programs.
- Good logging + terminal output.
- User can create own scripts and can use later on based on requirements.
- Self manageable.
- One centered configuration file for whole application.
- One place to store global variables and environment settings.
- For multiple scripts not creating multiple processes. Working as single process.
- Can define database connectivity easily for all types of database connections.
- Automatic log creation for process / jobs. We can also use already available program for logging like log4sh.
- Modularity but running on single processes.
- Can use any UNIX scripting language for whole application.
- Create own define entry point for starting application.
- Debugging will be easy.
- Can create a workflow for shell script with mix of other language.
- Can create propriety applications.
- Can create one simple compiler, which checking all commands before running application by defining new core engine methond.
- Create own library and register into Framework core engine.
- Create a testing suit.

V. CONCLUSION

After looking into above structure in detailed and example, we can create any app in this framework using any scripting language. Using this framework for development of application is easier to understand, create and integrate into a structured way. User able to use any UNIX scripting language and auto logging with debug feature, making it more usable. This framework can also be used into creating testing scenarios (test cases) because using configuration and XML file we can easily understand the workflow and running sequence with execution scope. Overall we can say using this framework in both forward engineering and backward engineering, will be beneficial for any programmer. Later on we can also use this framework for automatically script creating utility, means using some NLP tool we can extract task from statement and after we can create script based on task – script mapping

ACKNOWLEDGMENT

In the above example section, using a small program to show the processes. After using some strong programming language for the framework core engine, we can make this system more clear and powerful.

REFERENCES

[1] Mastering Unix Shell Scripting: Bash, Bourne, and Korn Shell Scripting for Programmers, System Administrators, and Unix Gurus, by Randal K. Michael
[2] Perl Programming for the Absolute Beginner by Jerry Lee Ford
[3] Python 3 for Absolute Beginners by Tim Hall, J-P Stacey
[4] http://code.google.com/p/shesfw/ , Shell Script Framework tool.
[5] http://leon.vankammen.eu/shellscript-framework
[6] http://en.wikipedia.org/wiki/Shell_script