Automated testing using Unix Shell Scripting

Software Testing Conference, Organized by QAI, India
Prepared by Prasad Patwa and Aniruddha Patwardhan

Abstract 1
Unix Introduction 2
  Unix architecture diagram 3
Advantages of using Shell for test automation on Unix 3
Some useful shell commands for test automation 4
Interactive Application testing using expect 6
Executing shell scripts on Windows using Cygwin 6
When not to use shell scripts for automated testing 6
References 7
Appendix A 8
  Sample bash script to automate installation testing 8
Appendix B 12
  Interactive Application testing using expect sample script 12

Abstract

Testing is a critical part of software development - and Shell, which is already part of Unix can help you do it quickly and easily. This paper, explains advantages of Unix shell scripting for test automation on Unix, useful shell commands for test automation, testing interactive application using expect, porting shell scripts on Windows, with some examples of automated test scripts developed using Shell Scripts.

Unix Introduction

Unix is one of the most popular operating systems, It has many advantages like.

- Multitasking: Unix is designed to do many things at the same time; e.g., printing out one file while the user edits another file. This is a major feature for users, since users don't have to wait for one application to end before starting another one.
- Multiusers: The same design that permits multitasking permits multiple users to use the computer. The computer can take the commands of a number of users -- determined by the design of the computer -- to run programs, access files, and print documents at the same time.
- Stability: Robustness and Stability is one of the design goals of Unix and its natively stable. Unix doesn't need to be rebooted periodically to maintain performance levels. It doesn't freeze up or slow down over time due to memory leaks and such. Continuous up-times of hundreds of days (up to a year or more) are not uncommon. Therefore requires less administration and maintenance.
- Performance: Unix provides persistent high performance on workstations and on networks. It can handle unusually large numbers of users simultaneously. . Also we can tune the unix systems in a better way to meet our performance needs ranging from embedded systems to SMP systems.
- Compatibility - Unix can be installed on many different types of machines, including main-frame computers, supercomputers and micro-computers. . Linux-A popular variants of Unix run on almost 25 architectures including Alphs/VAX, intel, PowerPC etc. Unix also is compatible with windows for file sharing etc via smb(samba file system) and NFS(Network File system)
- Security: Unix is one of the most secure operating systems. “Firewalls” and flexible file access permission systems prevent access by unwanted visitors or viruses.

Unix is mostly used for high performance server applications. Unix is the leader in serving the Web. About 90% of the Internet relies on Unix operating systems running on Apache, the world's most widely used Web server. With development of Linux it is also being used for desktop applications, and embedded systems.

Today, the combination of inexpensive computers and free high-quality Linux operating system and software provide incredibly low-cost solutions for both basic home office use and high-performance business and science applications.

With growth in development of Unix applications and the criticality of these application Testing of Unix application is also becoming critical.

With so many variants of Unix and increasing build frequency, testing effort has increased. Testing if done manually will be effort intensive and time consuming.

Automation of testing activities will help reduce the testing time and also increased the reliability of testing.

There are many testing tools in the marketplace that offer a lot of functionality to help with the testing efforts. However, they need to be obtained, installed, and configured,

which could take up valuable time and effort. Also most of the tools are windows specific and offer limited help to automating Unix side test efforts.

Shell which is a part of Unix already provides lot of powerful commands which can be used for automating the testing efforts.

Unix architecture diagram

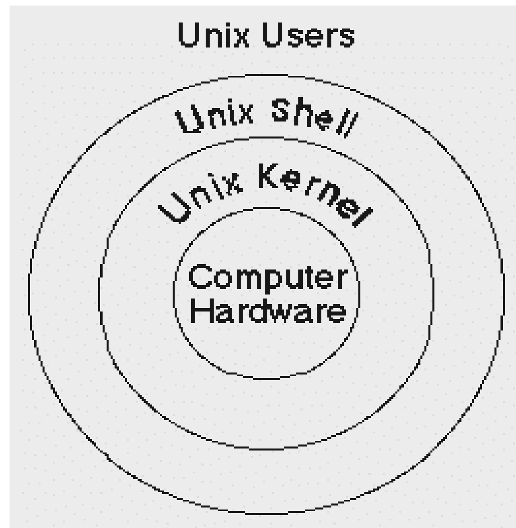

Fig: Simplified View of Unix Architecture

Shell, which is the 'command interpreter' for Unix systems, resides at the base of most of the user level Unix programs. All the commands invoked by us are interpreted by shell and it loads the necessary programs into memory. Thus being a default command interpreter on Unix makes shell a preferred choice to interact with programs and write glue code for test scripts.

Advantages of using Shell for test automation on Unix

Following are some of the advantages of using Shell for test automation on Unix,

- Free: Most of the popular shells are free and open source (GPL'ed) no additional cost. No Additional software required: All the Unix systems have a default shell already installed and configured (bash/ksh/csh). So no need to spend extra time getting it ready. Thus shell is something native to Unix systems and a native always understands the problems pretty well and help solving it.
- Powerful: Bash provides plenty of programming constructs to develop scripts having simple or medium complexity.
- Extensible: We can extend the shell scripts by using additional useful commands/programs to extend the functionality of the scripts
- We can write shell scripts using default editors available (vi, emacs etc) and can run and test it. No specialized tool is needed for the same.
- Can even generate color-highlighted reports of test case execution, which is of great help.

- Portability: Shell scripts are portable to other Unix platforms as well to Windows via Cygwin. Cygwin which is a shell on windows allows us to execute shell scripts on windows as well

Some useful shell commands for test automation

For testing we have to mostly do test setup, test procedure steps, validation of actual result with expected result, clean up steps to bring the application back to original state, scheduling a test, prepare test results log, and report the test results. Shell has many commands, which can help achieve automation of these test activities.

Following are some useful Unix Shell commands for automation,

1. Verification and setup testing:

When we want to test for installation/ uninstallation etc we can effectively use the file verification functionality of the shell.

```
e.g. if [ -d <dirname> ];
    then
                echo "dir exist"
    else
                echo "no such dir"
    fi
```

On similar lines we can use

- -f to check whether a file exist
- -r to check whether a file is readable
- -w to check whether a file is writeable
- -x to check whether a file is executable

We can also invoke external commands and check for their return code for success/failure of execution using predefined variable $?

e.g.

```
    some command
    if [ $? -eq 0 ];
    then
                echo "successfully executed"
    else
                echo "failed while executing"
    fi
```

Also availability of common looping constructs like 'for' and 'while' make shell obvious choice to automate installation/ uninstallation testing, checking out whether commands/programs are executing successfully or not and functionality testing as well.

Most of the time we need to setup some environment variables, have some proper links (test environment) to set, this task can be automated using shell and is of great help.

2. Functionality testing:

Most of the programs in Unix generate some kind of file or update some existing files. Via shell we can use grep (Global Regular Expression Parser) to parse the file for required patterns and based on those validate the execution of the particular program.

Sell also support sed (stream editor) and awk to manipulate the streams and to verify that required patterns are available at required places, this helps test team in validating the application

Some more commands like 'cut', 'sort', 'xargs' helps us in validating the required functionality of the programs under test.

3. Maintainability and grouping:

Shell provides 'function' to logically group the test cases. Using functions we can have each function for a test case and then we can call these functions from main script. This provides lot of maintainability and grouping support for test automation.

Once we have a function for each test case, we can invoke required functions as per our requirements, thus can create test sets like sanity test run, full test run etc.

Shell also provide the functionality to check for the return code from the functions so we can execute the test cases in a controlled way based on success/failure of sanity/ base test cases.

4. Scheduling:

Most of the times requirements say we need to schedule the automation scripts for a particular time. We can use 'cron' and 'at' for scheduling the scripts.

Using 'cron' we can schedule the scripts to execute every five minutes, every hour, every midnight or practically at any particular time we want. cron is used for repeated scheduling.

We can use 'at' if we want to schedule the scripts at a particular time but want to execute it only once and not in a repeated way.

5. Reporting:

We can use 'echo' or 'echo -e'(enhanced capabilities) to generate results after executing test cases. We can also generate colorized result reports for better understanding.

We can also use redirection operators '>' to send the output of the program/ script to some file and '>>' operator to append the output to some existing file. This helps us in generating reports as well.

We can also use following commands
'tee' - this command is useful for reading from a std input and writing it to both std output as well as file. This can be used in a effective way to generate reports on the console at the same time save them to a file.

We can also mail the generated reports or errors encountered using the mail command.
mail -s <subject> $MAILTO < $FILE_TO_MAIL
FILE_TO_MAIL is the file containing the text of the mail with only '.' on the last line. 'mail' program can be used to mail this file to user specified in the MAILTO variable.

Sample installation testing test script is given in Appendix A. The test results generated by this test scripts are also given.

Interactive Application testing using expect

Expect is a program that talks to other interactive programs according to a script. We need to mention to expect what to expect from the program and what is the response we need to send.

When you write an expect script the output from your program is an input to your expect script and output of the expect script is input to your program. So now your expect script keep on expecting output from the program and keep on feeding input the interactive program, thus automating the interactive programs.

Expect is generalized so that it can interact with any of the user level command/ program. Expect can also talk to several programs at the same time. In general expect is useful for running any program, which requires interaction between user and the program. All that is necessary is the interaction can be characterized using a program.

Example of automation interactive programs using Expect is given in Appendix B

Executing shell scripts on Windows using Cygwin

Cygwin is a Linux like environment for windows. It consists of two parts

- A dll, cygwin1.dll which acts as a Linux emulation layer providing Linux API functionality.
- A collection of tools, which provide Linux look and feel.

Cygwin is available under GPL (GNU Public License) and is free software.
Cygwin gives us almost all standard unix shells (bash, ksh, csh etc) so you can run most of your scripts on windows as well.

Thus cygwin provides lot of portability to shell scripts.

When not to use shell scripts for automated testing

Its not a good idea to use shell scripts in following cases.

- Need to generate or manipulate graphics or GUI
- Need port or socket I/O
- Complex applications with type checking, function prototyping etc
- Need data structures like linked lists, trees etc.

If any of the above is true it’s a good idea to use more powerful languages like C, C++ or Perl/ Python for test automation

References

"Using Bash shell scripts for function testing" by Angel Rivera
http://www-106.ibm.com/developerworks/linux/library/l-bashfv.html

"Advanced Bash scripting guide from The Linux Documentation Project (TLDP)"

"Unix Shell Programming" by Yashavant Kanetkar, BPB Publication.

Appendix A

Sample bash script to automate installation testing

The program below automates the installation testing of yahoo messenger rpm, a commonly needed rpm, which is, not get installed by default. This also validates the installation and generates a test report.

Here we are grouping different test cases in function e.g. ym_install() – this functions tries to install the rpm and returns 0/1 based on whether its successful or not. Similarly ym_install_dir_validate() function validates the creation of required directories.

Now we invoke all this functions via a main script and check for validity of each test case. Finally we generate the test report about whether the tests are successful or not.

Bash Shell Script to automate installation testing of yahoo messenger
--

```
#!/bin/sh
# Script to install ym and validate its installation
SRC_RPM=$1
TARGET_DIR="/opt/ymessenger"
TARGET_FILE="$TARGET_DIR/bin/ymessenger"
YM_LIB="$TARGET_DIR/lib/libgtkhtml.so.6"

INSTALL_LOG="ym_install.log"

SETCOLOR_SUCCESS="echo -en \\033[1;32m"
SETCOLOR_FAILURE="echo -en \\033[1;31m"
SETCOLOR_WARNING="echo -en \\033[1;33m"
SETCOLOR_NORMAL="echo -en \\033[0;39m"

ym_install()
{
	echo -n "Installing RPM"

	rpm -Uvh --force $SRC_RPM > /dev/null
	if [ $? -eq 0 ]; then
		$SETCOLOR_SUCCESS
		echo -e "\t [OK]"
		$SETCOLOR_NORMAL
	else
		$SETCOLOR_FAILURE
		echo -e "\t [FAILED]"
		$SETCOLOR_NORMAL
	fi

	return 0
}
```

```
ym_install_dir_validate()
{
	echo -n "Checking for Directory $TARGET_DIR"

	if [ -d $TARGET_DIR ]; then
		$SETCOLOR_SUCCESS
		echo -e "\t [OK]"
		$SETCOLOR_NORMAL
	else
		$SETCOLOR_FAILURE
		echo -e "\t [FAILED]"
		$SETCOLOR_NORMAL
	fi

	echo -n "Checking for Directory $TARGET_DIR/bin"

	if [ -d $TARGET_DIR ]; then
		$SETCOLOR_SUCCESS
		echo -e "\t [OK]"
		$SETCOLOR_NORMAL
	else
		$SETCOLOR_FAILURE
		echo -e "\t [FAILED]"
		$SETCOLOR_NORMAL
	fi

	echo -n "Checking for Directory $TARGET_DIR/lib"

	if [ -d $TARGET_DIR ]; then
		$SETCOLOR_SUCCESS
		echo -e "\t [OK]"
		$SETCOLOR_NORMAL
	else
		$SETCOLOR_FAILURE
		echo -e "\t [FAILED]"
		$SETCOLOR_NORMAL
	fi


	return 0
}

ym_install_file_validate()
{
	echo -n "Checking for file permissions on $TARGET_FILE"

	if [ -x $TARGET_FILE ]; then
		$SETCOLOR_SUCCESS
		echo -e "\t [OK]"
		$SETCOLOR_NORMAL
	else
```

```
		$SETCOLOR_FAILURE
		echo -e "\t [FAILED]"
		$SETCOLOR_NORMAL
	fi


	return 0
}

ym_install_lib_validate()
{
	echo -n "Checking for library of $TARGET_FILE"

	if [ -x $YM_LIB ]; then
		$SETCOLOR_SUCCESS
		echo -e "\t [OK]"
		$SETCOLOR_NORMAL
	else
		$SETCOLOR_FAILURE
		echo -e "\t [FAILED]"
		$SETCOLOR_NORMAL
	fi

	return 0
}

echo -e "\t\tAutomated testing of YM install procedure"
echo -e "\t\t-----------------------------------------"

ym_install
if [ $? -ne 0 ]; then
	echo "Error: Install Failed."
	echo "Error: Aborting Remaining test cases."
	exit 1
fi

echo " "
ym_install_dir_validate
if [ $? -ne 0 ]; then
	echo "Error: Check for directory failed."
	echo "Error: Aborting Remaining test cases."
	exit 1
fi
echo " "

ym_install_file_validate
if [ $? -ne 0 ]; then
	echo "Error: Check for files and permissions failed."
	echo "Error: Aborting Remaining test cases."
	exit 1
fi
```

```
echo " "

ym_install_lib_validate
if [ $? -ne 0 ]; then
        echo "Error: Install library validation failed"
        echo "Error: Aborting Remaining test cases."
        exit 1
fi

echo " "
echo -e "Summary:"
echo -e "--------"
echo -e "YM Install Succesful."
echo -e "All Sanity test cases executed successfully without any error."

exit 0
```

Report Generated by the automation script
--

```
            Automated testing of YM install procedure
            ----------------------------------------------------
        Installing RPM                                                    [OK]
        Checking for Directory /opt/ymessenger                            [OK]
        Checking for Directory /opt/ymessenger/bin                        [OK]
        Checking for Directory /opt/ymessenger/lib                        [OK]
        Checking for file permissions on /opt/ymessenger/bin/ymessenger   [OK]
        Checking for library of /opt/ymessenger/bin/ymessenger            [OK]

        Summary:
        -------------
            YM Install Succesful.
            All Sanity test cases executed successfully without any error.
```

Appendix B

Interactive Application testing using expect sample script

Program to automate: (test.sh)

```
#!/bin/sh
echo "Enter name"
read name

echo "Hi $name. Good morning"
```

Expect script to automate:

```
#!/usr/bin/expect -f
spawn ./test.sh

expect "Enter name"
send "anix\n"

interact
```