CSE2031 Software Tools - UNIX scripting

Summer 2010

Przemyslaw Pawluk

Department of Computer Science and Engineering
York University
Toronto

July 20, 2010

Notes

Table of contents

Notes

Bourne shell (sh)

sh - basics

This shell was developed by Stephen Bourne, of AT&T Bell Laboratories, and was released in 1977 in the Version 7 Unix release distributed to colleges and universities. It remains a popular default shell for Unix accounts. The binary program of the Bourne shell or a compatible program is located at /bin/sh on most Unix systems, and is still the default shell for the root superuser on many current Unix implementations. Its command interpreter contained all the features that are commonly considered to produce structured programs. Although it is used as an interactive command interpreter, it was always intended as a scripting language.

Notes

sh features

- Scripts can be invoked as commands by using their filename
- May be used interactively or non-interactively
- Allow both synchronous and asynchronous execution of commands
- Supports input and output redirection and pipelines
- Provides a set of builtin commands
- Provides flow control constructs, quotation facilities, and functions.
- Type-less variables
- Provides local and global variable scope
- Scripts do not require compilation before execution
- Does not have a goto facility, so code restructuring may be necessary

Notes

sh features cont.

- Command substitution using back quotes: ‘command‘.
- Here documents using << to embed a block of input text within a script.
- `for - do - done` loops, in particular the use of `$*` to loop over arguments.
- `"case - in - esac"` selection mechanism, primarily intended to assist argument parsing.
- sh provided support for environment variables using keyword parameters and exportable variables.
- It contains strong provisions for controlling input and output and in its expression matching facilities.

Notes

bash - Bourne-again shell

bash - basics

Bash is a free software Unix shell written for the GNU Project. Its name is an acronym which stands for Bourne-again shell. Bash was created in 1987 by Brian Fox.
Bash is a POSIX shell with a number of extensions. It is the shell for the GNU operating system from the GNU Project. It can be run on most Unix-like operating systems. It is the default shell on most systems built on top of the Linux kernel as well as on Mac OS X and Darwin. It has also been ported to Microsoft Windows using Subsystem for UNIX-based Applications (SUA), or POSIX emulation provided by Cygwin and MSYS. It has been ported to MS-DOS by the DJGPP project and to Novell NetWare.

Notes

bash - features

- The Bash command syntax is a superset of the Bourne shell command syntax.
- The vast majority of Bourne shell scripts can be executed by Bash without modification, with the exception of Bourne shell scripts stumbling into fringe syntax behavior interpreted differently in Bash
- Bash command syntax includes ideas drawn from the Korn shell (ksh) and the C shell (csh) such as
  - command line editing
  - command history
  - the directory stack
  - the `$RANDOM` and `$PPID` variables
  - POSIX command substitution syntax `$()`
- When used as an interactive command shell and pressing the tab key, Bash automatically uses command line completion to match partly typed program names, filenames and variable names.

Notes

bash features cont.

- can perform integer calculations without spawning external processes
- Bash uses the `(())` command and the `$(())` variable syntax for this purpose.
- Bash syntax simplifies I/O redirection (For example, Bash can redirect standard output (stdout) and standard error (stderr) at the same time using the &> operator)
- Bash function declarations (using the key word 'function') are not compatible with Bourne/Korn/POSIX/C-shell scripts. Due to these and other differences, Bash shell scripts are rarely runnable under the Bourne or Korn shell interpreters unless deliberately written with that compatibility in mind
- Bash supports here documents just as the Bourne shell always has. However, since version 2.05b Bash can redirect standard input (stdin) from a "here string" using the <<< operator.
- Bash 3.0 supports in-process regular expression matching using a syntax reminiscent of Perl.

Notes

Brace expansion

Def

Brace expansion is a feature, originating in csh, that allows arbitrary strings to be generated using a similar technique to filename expansion. However the generated names need not exist as files. The results of each expanded string are not sorted and left to right order is preserved

bash-example

```
1 bash-3.00$ echo a{p,c,d,b}e
2 ape ace ade abe
```

sh

```
1 red 303%echo a{p,c,d,b}e
2 a{p,c,d,b}e
```


Notes

More information

- man pages (`man sh`, `man bash` ...)
- sh documentation - `http://steve-parker.org/sh/bourne.shtml`
- bash home - `http://www.gnu.org/software/bash/bash.html`
- comparison of different shells - `http://en.wikipedia.org/wiki/Comparison_of_command_shells`

Notes

functions in shell

functions

The original version of the Bourne shell didn't have functions. If you wanted to perform an operation more than once, you either had to duplicate the code, or create a new shell script. The Bourne shell solved this problem with the concept of functions.

```
1  inc_A() {
2  # Increment A by 1
3      A=`expr $A + 1`
4  }
5  A=1
6  while [ $A -le 10 ]
7  do
8      echo $A
9      inc_A
10 done
```


Notes

Passing values by name

You can pass names of variables to functions. However, this adds a lot of complexity. You must bypass the normal shell evaluation of variables. Also, strings like "$$" have special meanings.

Increment a specified variable

```
1  #!/cs/local/bin/sh
2  inc() { eval $1=`expr $$1 + 1`; }
```


Notes

Exiting from a function

There are three ways of exiting a function

- Normally, the function returns with the exit status of the last command
- If you want to control explicitly the value, the Bourne shell has a special command called `return` that sets the status value to the value specified.
- If you execute an `exit` command inside a function it aborts the script, and passes the value to the calling script

Notes

Arguments

How to check no of arguments?

Let's say you have a shell script with three arguments. There are many ways to solve this problem. One way to make sure your script will work without the right number of arguments is to use default values for the variables or you can use the form that reports an error if an argument is missing.

Notes

Examples

Default

```
1 #!/cs/local/bin/sh
2 arg1=${1:-a.out}
3 arg2=${2:-`pwd`}
4 arg3=${3:-$HOME}
5 mv $arg2/$arg1 $arg3
```

Request

```
1 #!/cs/local/bin/sh
2 file_to_be_moved="$1"
3 arg1=${file_to_be_moved:?"filename missing"}
4 arg2=${2:-`pwd`}
5 arg3=${3:-$HOME}
6 mv $arg2/$arg1 $arg3
```


Notes

Examples

Request all

```
#!/cs/local/bin/sh
arg1="a.out"
arg2=`pwd`
arg3=$HOME
if [ $# -eq 3 ]
then
        arg1="$1";
        arg2="$2";
        arg3="$3";
else
        echo you must specify exactly 3 arguments
        exit 1
fi
echo $arg1 $arg2 $arg3
mv $arg2/$arg1 $arg3
```


Notes

Optional arguments

```
#!/cs/local/bin/sh
usage() {
echo usage: `basename $0` '[-a] [-o file][file ...]' 1>&2
exit 1
}
a= o=
while :
do
    case "$1" in
-a) a=1;;
-o) shift; o="$1";;
--) shift; break;;
-*) usage "bad argument $1";;
*) break;;
    esac
    shift
done
```


Notes

Example 1 - find a user in a CSV file

You're suppose to find a user name, surname and id by name which is passed as a script parameter.

Notes

Example 2 - find all names of users that have borrowed a book

You're suppose to find all users' names and surnames that have borrowed a book. Book id is passed as a script parameter.

Notes

Notes

Notes