

Shell Scripting



September 6, 2005

What's a Shell?



- ✓ The most basic user interface.
- ✓ “Command Line” - Windows
- ✓ “DOS Window” - Windows
- ✓ “Terminal” - Unix
- ✓ Sometimes “Console” - Unix

Common Unix Shells

- ✓ sh - the bourne shell
 - ✓ Stephen Bourne - AT&T Bell Laboratories
- ✓ csh - the C programmer's shell
 - ✓ Bill Joy - University of California
- ✓ bash - "Bourne Again Shell"
 - ✓ Richard Stallman - Free Software Foundation
- ✓ tcsh - an updated csh
 - ✓ 't' is for TENEX - a old system eventually bought by DEC in the late 70's.
- ✓ ksh - the korn shell
 - ✓ David Korn - AT&T Bell Laboratories

Why Your Default Shell is tcsh



- ✓ tcsh has very flexible completion and variable substitution routines built-in.
 - ✓ Examples: tab completion, spelling correction, variables...
- ✓ The Sun Solaris operating system defaults to it.

.cshrc And .bashrc Files



- ✓ Modify your shell by editing your .cshrc (tcsh) or .bashrc (bash)
- ✓ Both are shell scripts!
- ✓ \$PATH
- ✓ \$LD_LIBRARY_PATH
- ✓ aliases

Why Shell Script?



- ✓ It's the simplest form of programming.
- ✓ Shell scripting is exactly the same as you use with an interactive shell session.
- ✓ Greatly reduces time spent doing repetitive commands.
- ✓ EASY!

Why tcsh is Bad for Scripting



- ✓ Since most of the Unix system is built upon the sh shell, bash is a more sensible choice.
- ✓ “csh Programming Considered Harmful”
<http://www.tac.nyc.ny.us/mirrors/tcsh-book/csh-whynot>

Getting Started with bash

- ✓ At any Unix prompt, type 'bash'.
- ✓ To start programs, do it like you would in tcsh.
- ✓ Differences between tcsh and bash are mainly in variable declaration, I/O

```
bash-2.95b $
```

```
$ ls
```

```
bin
```

```
code
```

```
Documents
```

```
tmp
```

```
$ emacs
```

Variables

- ✓ Any word can be used, but typically variables are written in UPPER CASE.
- ✓ After declaring the variable, you refer to it by using a dollar sign (\$).

```
$ TEST=1
$ echo $TEST
1
$ echo $SHELL
/bin/bash
$ printenv
TEST=1
SHELL=/bin/bash
```

Flow Control



- ✓ if; then; else; fi
- ✓ while; do; done
- ✓ for; do; done

```
if [ "TEST" = "1" ]; then
    <do this>
else
    <do this instead>
fi
```

```
for i in 1,2,3; do
    <do this to $i>
done
```

Putting This into a Script

- ✓ Begin with sha-bang
 - ✓ This identifies which shell you will be using.
- ✓ Comment heavily
 - ✓ Use a # sign anywhere.
- ✓ Execute by bash-ing it
 - ✓ \$ /bin/bash script.sh
 - or -
 - ✓ \$ chmod a+x script.sh
 - \$./script.sh

```
#!/bin/bash
# My first script
# Usage: script.sh

echo "Hello, World"

for WIDGETS in a,b,c; do
    echo $WIDGETS
done

echo "Hit enter to finish"
read $VAR
```

Interacting With a Script

- ✓ `read VARIABLE`
 - ✓ Assigns any input to `$VARIABLE`
- ✓ **Script arguments**
 - ✓ Allows you to pass extra options to a script when you run it.

```
#!/bin/bash
# My 2nd script
# Usage: script2.sh

echo "type something:"
read VAR
echo "you typed:"
echo $VAR

echo $1           # Prints the 1st argument
echo $2           # Prints arg #2
echo $@           # Prints all args
echo $#           # Prints the number of args
echo $?           # Prints the status of the last
                  # command
```

` VS ' VS “



- ✓ These have different meanings:
- ✓ ' is a strong quote
 - ✓ Use for literal interpretation of a command
- ✓ “ is a weak quote
 - ✓ Use for partial interpretation of a command
- ✓ ` is used for command substitution

Example

```
for i in `seq 1 3`; do  
    echo $i  
done
```

1
2
3

```
echo $PATH  
echo '$PATH'  
echo "$PATH"
```

/bin:/usr/bin
\$PATH
/bin:/usr/bin

I/O



- ✓ > redirects output to file
- ✓ >> appends output to file
- ✓ < get input from a file
- ✓ | (pipe) feeds output of left command to right command

Good Books on Scripting



- ✓ Newham, Rosenblatt -
Learning the bash Shell (1998)
- ✓ Taylor -
Wicked Cool Shell Scripts (2004)