Bash 101
Intro to Shell Scripting

PLUG North 2011-02-08
PLUG West 2011-02-21

Updated: 2011-02-22

JP Vossen, CISSP
bashcookbook.com

http://www.jpsdomain.org/public/2011_bash_101.pdf
http://www.jpsdomain.org/public/2011_bash_101.odp

Agenda

- What is a “shell” and a “shell script?”
- Why should I care?
- How do I get started?
- Prompts, positional parameters & STDIO
- Anatomy of 'cdburn'
- Programming bash
- What did we miss?
- What about Windows?
- What next?
- URLs, Wrap up and Q&A

What is a “shell?”

- A program that provides an interface to an operating system (which is itself an interface to the hardware)
- May be CLI or GUI
  - CLI = command line interface
  - GUI = graphical user interface
- May have more than one available
  - Bourne (sh), Bourne Again Shell (bash), Korn (ksh)
  - zsh, fish, csh, tcsh, many, many others
  - CDE, Gnome, KDE, Presentation Manager, Workplace Shell, many, many others

What is a “shell script?”

- Fundamentally just a list of commands to run
  - May use arguments and variables various control logic and arithmetic to figure out what or run when
  - bash is integer only, other shells may not be
- Plain text file
- Used on CLI only
- Builds on:
  - The “Unix” tool philosophy
  - The “Unix” everything-is-a-file philosophy

Why should I care?

- You can write new commands!
  - Save time & effort and make life easier
  - E.g., if you always type in four commands to accomplish some task, type them once into an editor, add a “shebang” line and comments, save and set execute permissions. You now have a shell script!
- Automation
  - cron
- Consistency & Reliability
- (Process) Documentation
- One-liners

How do I get started?

- Fire up an editor

  -
    ```
    #!/bin/bash -
    echo 'Hello world, my first shell script!'
    ```
  - `chmod +r script`
- bash 'help' command!
  - 'help set' vs. 'man set'
- Most of a Linux system is run by shell scripts. They are everywhere, find some and read them.
  - Everything in /etc/init.d/

  -
    ```
    for i in /bin/*; do file $i | grep -q 'shell script' && echo
    $i; done                    # You will be surprised!
    ```

A Word About Prompts

- http://www.tldp.org/HOWTO/Bash-Prompt-HOWTO/index.html
- PS1 is the interactive prompt (default '\s-\v\$ ', varies by distro)
  - PS1='\n[\u@\h:T\l:L$SHLVL:C\!:J\j:\D{%Y-%m-%d_%H:%M:%S_%Z}]\n$PWD\$ '
  - [user@hostname:T0:L1:C924:J0:2011-02-08_17:42:33_EST] /home/user/Documents/Presentations$
- PS2 is the continuation prompt (default is '> ' which is OK)
  - PS2='>'
- PS3 is the 'select' prompt (default of '#? ' is kinda useless)
  - PS3=''
- PS4 is the debug (trace) prompt (default of '+ ' is kinda useless)
  - PS4='+xtrace $LINENO: '

Positional Parameters

- “Main” script:

```
    $0      $1  $2  $3
myscript  foo  bar  baz
```

- $# = number of parms
- $* = “$1 $2 $3”   # a single string of all parms, separated by first character of $IFS (Internal Field Separator)
- “$@” = “$1” “$2” .. “$N”   # For re-use later
- Reset inside a function
  - $1 = first arg to function, not script
  - But use $FUNCNAME instead of $0

Standard Input, Output & Error

- http://en.wikipedia.org/wiki/Standard_streams
- STDIN = standard input, usually from the keyboard or another program via a pipeline or redirection
- STDOUT = standard output, to terminal, pipeline or redirection
  - echo 'Hello World!'
- STDERR = standard error, to terminal, pipeline or redirection but allows errors to be seen even if STDOUT is piped or redirected
  - echo 'World Hello!' >&2

Anatomy 1

- “Shebang” line → /bin/sh -ne /bin/bash

```
#!/bin/sh -
#!/bin/bash -
#!/usr/bin/env bash
```

- Comment line

  # *name--description*

```
# cdburn--Trivially burn ISO images to disc
```

- Version control line (optional, depends)

```
# $Id$
VERSION='$Version: 1.1 $'  # CVS/SVN
# VERSION='ver 1.2.3'        # Hard-code
```

Anatomy 2: Usage

```
if [ "$1" = "-h" -o "$1" = "--help" -o -z "$1" ]; then
    cat <<-EoU
        $0 $VERSION
        Trivially burn ISO images to disc
        Usage: $0 </path/to/iso>
        e.g. $0 /home/jp/CD-image/image.iso
    EoU
    exit 1      # or 'exit 0'?
fi
```

Anatomy 3: Sanity Checks

```
speed=''           # Use burner default (2x ' not “)

# Make sure we have a burner

if   [ -x /usr/bin/wodim ]; then

    # Debian, Ubuntu
    CDBURNER='/usr/bin/wodim'

elif [ -x /usr/bin/cdrecord ]; then

    # CentOS, etc.
    CDBURNER='/usr/bin/cdrecord'

else

    echo "FATAL: Can't find wodim or cdrecord!  Is either
installed?"
    exit 1

fi
```

Anatomy 4: guts

```
ISO="$1"

[ -r "$ISO" ] || {
    echo "FATAL: ISO '$ISO' not found or not readable!"
    exit 2
}

PS4=''   # That is ' and ', not "
set -x    # “debug”; will display cmd then run it

$CDBURNER -v -eject -dao $speed padsize=63s -pad "$ISO"
```

Notice...

- The code (“guts”) that actually does the work is usually only a tiny amount of code.
- 70-95% of the code is usually the “user interface:”
  - Prevent mistakes
  - Give useful feedback
- Code for GUI's (Graphical User Interfaces) is even worse; it's larger and almost all of the code is “interface” with only a tiny bit being guts.

“Programming” bash

- programming language
- basic operation is invocation = you run stuff
- variables
  - integers
  - strings
  - arrays
- control structures
  - Branching / conditionals
  - looping

debugging

- PS4='+xtrace $LINENO: '
  - First character is duplicated to show nesting level, that's why I have '+' there
  - $LINENO should be in the default PS4 prompt!

- bash -n path/to/script   # gross syntax check
- bash -x path/to/script   # debug

- set -x & set +x   # debug on / off
- set -v & set +v   # verbose on / off

Quotes

- The shell re-writes the line
- White space is a delimiter!
- Quoting
  - Use ' ' unless you are interpolating $variables, then use " "
  - echo 'foo'
  - echo "$foo"
  - grep 'foo' /path/to/file
  - grep "$regex" /path/to/file
- Except when it's not. Can make your head hurt.

Variables

- USE GOOD NAMES!!!
- No $ or spaces around = when assigning:
  ```
  foo='bar'
  foo="bar$baz"
  ```
- $ when referencing value:
  ```
  echo "$foo"
  ```
- Append:
  ```
  foo="$foo bar"
  ```
- Needs ${} if variable followed by [a-zA-Z_0-9]
  ```
  foo="foo $bar baz"    # OK
  foo="foo${bar}baz"    # $bar needs ${}
  ```

Command Substitution

- Old way (backticks):
  ```
  ` `
  ```
- New way, easier to read and nest:
  ```
  $( )
  ```
- Example:
  ```
  lines_in_file=$(wc -l $file | awk '{print $1}')
  ```
- The effect is to pull outside data into your script, which is terribly useful.

I/O Redirection

- command > output
  ```
  ls >  mydir.txt     # create or truncate
  ls >> mydir.txt     # create or append
  ```
- command < input
  ```
  wc < mydata
  ```
- command1 | command2   # AKA pipeline
  ```
  ls | wc -l
  ```
- `cmd > outfile 2> errfile`
- `cmd > logfile 2>&1   # or just >&`
- `cmd 2>&1 | next command`

If .. then .. else .. fi

-
  ```
  if [ "$1" = "-h" -o "$1" = "--help" -o -z "$1" ]; then
      stuff
  elif grep -q "$pattern" "$file"; then
      stuff
  else
      stuff
  fi
  ```

-
  ```
  grep -q "$pattern" "$file" && {
      echo "Found '$pattern' in '$file'!"
      exit 0
  } || {
      echo "Did not find '$pattern' in '$file'!"
      exit 1
  }
  ```

for .. do .. done

- Execute commands for each member in a list
  - for i in /bin/*; do file $i | grep -q 'shell script' && echo $i; done
  - for i in /bin/*; do
        file $i | grep -q 'shell script' && echo $i
    done

  - for octet in $(seq 1 10); do host 192.168.1.$octet; done | grep -v 'NXDOMAIN)$'
  - for partition in 1 2 3; do mdadm --add /dev/md$partition /dev/sda$partition; done
  - for file in *.JPG; do echo mv $file ${file/JPG/jpg}; done

case .. esac

- “Execute commands based on pattern matching”

```
case "$HOSTNAME" in
    drake* ) speed='speed=24' ;; # GCC-4244N, Write: 24x CD-R, Rewrite: 24x CDRW, Read: 8x DVD ROM, 24x CDROM
    ringo* ) speed='speed=48' ;; # Man.Part# : G9P3H / Dell Part# : 318-0037
    * )      speed='speed=4'  ;; # Ancient default, but it worked
esac
```

select .. done

- Sort-of trivially create a user menu
  - “Sort-of” because you need to get your logic right
  - Trivial example without any error or other checking or an “exit” option:

```
PS3='Choose a file: '
select file in $dir/*; do
    echo "$file" && break
done
```

docs

- “here-document”
  - Must use TAB, not space to indent when using '<<-'!!!
  - cat <<EoF   cat <<-EoF
  - cat <<'EoF'   cat <<-'EoF'
- Comments
  - May be stand-alone or in-line after code
    - # Stand-alone comment
    - ls -la /root # Long list including hidden files of /root
- In-line POD (Perl's Plain Old Documentation)
  - pod2html, pod2latex, pod2man, pod2text, pod2usage
  - Use a NoOp + here-document
    - : <<'POD'

functions

- There's a bunch of ways to declare them, I like:
  - function foo {
    ```
    function foo {
            <code goes here>
    }
    ```
- $1, $2 .. $N get reset inside the function
  - Use $FUNCNAME instead of $0
  - Can also use 'local' keyword for scope
- CAN'T pass values back out like you'd expect!!!
  - Either set GLOBAL variables
    - Except watch out for subshells (including '|')!!!
  - OR output results and call function in a $()

Function _choose_file

```
#++++++++++++++++++++++++++++++++++++++++
# “Return” the file name chosen (not for production use)
# Called like:  file=$(_choose_file <dir>)
function _choose_file {
    local dir="$1"
    PS3='Choose a file: '
    select file in $dir/*; do
        echo "$file" && break
    done
} # end of function _choose_file
```

Revision Control

- Out of scope here, except that you want some.
- Lots of resources out there.
    - http://www.jpsdomain.org/public/PANTUG_2007-06-13_appd=Revision_Control=JP.pdf
- Trivial case:
    - aptitude install bzr
    - cd /path/to/scripts
    - bzr init
    - bzr add *
    - bzr ci

What did we miss?

- Well, almost everything, entire books have been written, 1 hour isn't going to cover it.

- for ((expr1 ; expr2 ; expr3)) ; do list; done
- while list; do list; done
- until list; do list; done
- Pattern Matching:
  - ${variable#pattern} ${variable##pattern}
  - ${variable%pattern} ${variable%%pattern}
  - ${variable/pattern/string} ${variable//pattern/string}

What else did we miss?

- String Operations:
  - ${variable:-word}   # Return a default value
  - ${variable:=word}   # Set a default value
  - ${variable:?word}   # Catch undefined vars
  - ${variable:+word}   # Test existence
  - ${variable:offset:length}   # Substrings
- Aliases (& \unalias)
- Lots, lots, lots more...

What about Windows?

- bash comes on a Mac, but not on Windows.
- Windows 'cmd.exe' is actually much more powerful than most people realize, but it still pales in comparison to any decent Unix/Linux shell.
  - http://www.jpsdomain.org/windows/winshell.html
- Use Cygwin: http://www.cygwin.com/
- Use the UnxUtils: http://unxutils.sourceforge.net/
- Use the GNU Win32 ports: http://sourceforge.net/projects/gnuwin32/
- Use Perl, Python or some other tool
  - http://www.activestate.com/solutions/perl/, etc.

What next?

- Books
  - *Learning the bash Shell*
  - *Bash Cookbook*
  - *Classic Shell Scripting*
- Web
  - http://www.bashcookbook.com/bashinfo/
  - Google
  - Everywhere
- Revision Control
  - Bazaar (BZR), git, Subversion (SVN), many others
  - Avoid CVS if possible, it's too old and crufty

URLs, Wrap-up and Q&A

- URLs:
  - TONS of resources: http://www.bashcookbook.com/bashinfo/
  - These slides: http://www.jpsdomain.org/public/2011_bash_101.pdf
    http://www.jpsdomain.org/public/2011_bash_101.odp
  - Bash vs. Dash: http://princessleia.com/plug/2008-JP_bash_vs_dash.pdf and
    aptitude install devscripts then use *checkbashisms*
  - The sample script: http://www.jpsdomain.org/public/cdburn
  - STDIN, STDOUT, STDERR: http://en.wikipedia.org/wiki/Standard_streams
  - Revision Control: http://www.jpsdomain.org/public/PANTUG_2007-06-13_appd=Revision_Control=JP.pdf
  - Windows Shell Scripting (cmd.exe): http://www.jpsdomain.org/windows/winshell.html
  - BASH Prompt HOWTO: http://www.tldp.org/HOWTO/Bash-Prompt-HOWTO/index.html
  - Cygwin: http://www.cygwin.com/
  - UnxUtils: http://unxutils.sourceforge.net/
  - GNU Win32 ports: http://sourceforge.net/projects/gnuwin32/
  - Win32 Perl http://www.activestate.com/solutions/perl/
- Questions?
- I'm on the PLUG list... jp@jpsdomain.org
- Some of these slides were adapted from 2007 Ubuntu Live presentation by Carl Albing & JP Vossen: “bash from beginner to power user”