COMP 175

System Administration and Security

UNIX SHELLS

UNIX Commands

- A command is a program which interacts with the kernel to provide the environment and perform the functions called for by the user.
- A command can be:
  - **a built-in shell command** 2 out of 3
  - **an executable shell script**
  - a source compiled, object code file
- **The shell is a command line interpreter**. The user interacts with the kernel through the shell. You can write ASCII (text) scripts to be acted upon by a shell.

UNIX Shells

- The shell sits between the user and the OS acting as a command interpreter
- Reads terminal input and translates the commands into actions taken by the system.
- Analogous to `command.com/cmd` in DOS/Windows
- System logon provides a default shell
- As shell starts it reads configuration files
  - set environment variables
  - command search paths
  - command aliases
  - executes any specified commands

UNIX Shell

UNIX – command line interface, called a shell

Bourne shell (sh) Protocol

- Shell starts up and initializes itself
- Shell presents a prompt character (% or $ sign) and waits for user to type a command line
- User types a command line – presses enter key
  - Shell extracts 1st word (name of program to run)
  - Shell searches for this program
  - If found, runs program, else error message
  - Shell suspends itself until program terminates
- Shell waits for another command

UNIX Shell

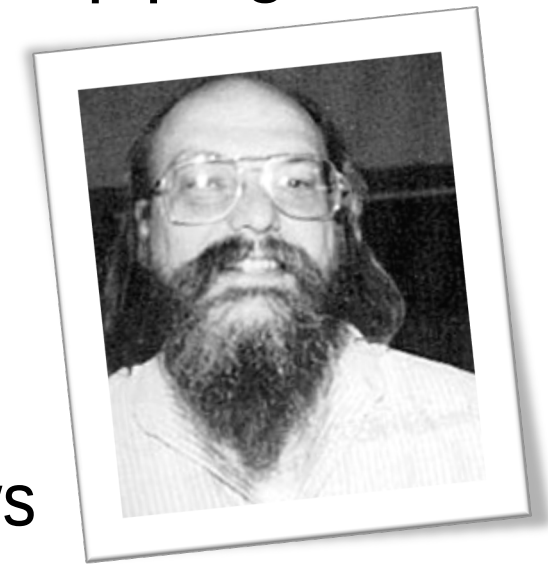
- Shell an ordinary user program – needs read/ write access, able to execute other programs
- Commands may take arguments, which are passed to the called program as character strings
- Shell does not have to open terminal, but it has access automatically to a:
  - file ‘standard input’ (for reading)
  - file ‘standard output’ (for writing normal output)
  - file ‘standard error’ (for writing error messages)
- Shell can redirect standard input/output to files
  - Example: `sort <in >out`

UNIX Shell

- Pipe symbol
  - Example: `grep ter *.t | sort >out`
    - All the lines containing the string “ter” in all the files ending in “.t”, sorted, written to file “out”
  - Possible pipeline – sequence of pipe symbols
- Single user can run several programs at once
  - Shell syntax to run program in background `&`
  - Example: `wc -l <a >b &`
- Shell scripts – files containing shell commands

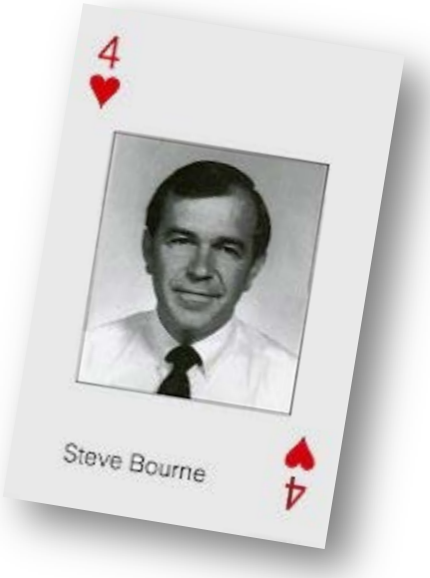
Thomson shell (sh)

- Ken Thompson, 1971
- First Unix shell in the first release of Unix
- Simple command interpreter
- Not really a scripting language
- Thompson shell syntax for redirection, piping
  - <
  - >
  - |
- adopted by most other Unix shells
- adopted by DOS, OS/2, and Windows
- also b, worked on regular expressions, ed, UTF-8



Bourne shell (sh)

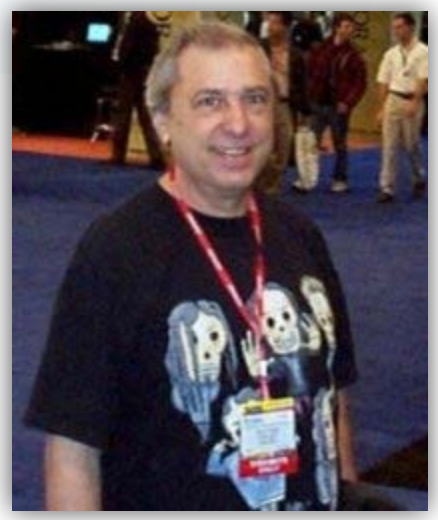
- Stephen Bourne, ATT&T Bell Labs (1977)
- /bin/sh - Replacement for Thompson shell
- interactive command interpreter
- command programming language.
- Released in 1977 as default Unix Version 7 shell
- Default shell for the root (superuser) account
- Descendants: ksh, rc, bash, dash
- Lacked history, aliases, job control



C Shell (csh)

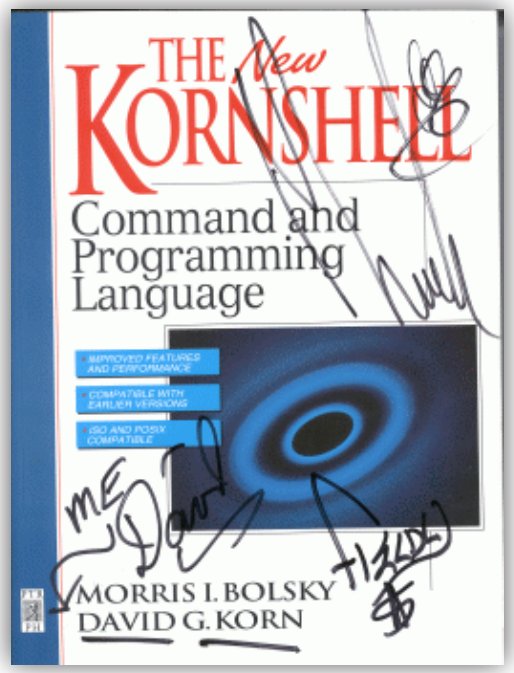
- Bill Joy for BSD, 1978
- Derived from Thompson shell (original sh)
- Syntax modeled after C
- Good job control features, history
- Default prompt is %
- tcsh – C shell (csh) with features, e.g., command-line editing
- Typical usage:
  - C Shell (csh) for interactive use (or tcsh)
  - Bourne shell (sh) for scripting

Korn Shell (ksh)



- David Korn, 1988 (ksh88), Bell Labs
  - Syntax of Bourne shell (sh)
  - Features of C Shell (csh)
  - Functionality of perl, awk, tcl
- Variants: dtksh (part of CDE), tksh (with Tk)
- Basis for POSIX shell
- Ksh93
- see: www.kornshell.com

Kornshell reference manual signed by members of KoRn



Bash Shell

- “Bourne-Again” shell
- GNU Project, 1987
- Superset of Bourne shell (sh)
- Features of C Shell (csh), and tcsh
- Default for most modern Linux distributions, Mac OS X, Cygwin
- Default prompt is $ (# for root user)
- “bashisms” – bash extensions that are not strictly POSIX compliant, may cause portability issues

- Ubuntu – see: https://wiki.ubuntu.com/DashAsBinSh

ash & dash

- ash - (Kenneth) Almquist shell
  - ash is clone of SVR4 Bourne shell - compliant
  - Small memory and space requirements - 92K
  - Often seen on embedded systems
    - Is the shell on the EndRun NTP servers
- dash - Debian version of ash
  - POSIX-compliant implementation of /bin/sh
  - Direct descendant of the NetBSD version of ash
  - Goal - to be as small as possible
  - No bashisms – features not in sh
  - Faster startup

Other Shells

- rc – replacement for sh on Plan 9
- esh – Easy Shell, Lisp based
- scsh – Scheme shell
- sash – Standalone shell, no reliance on external libraries
- zsh – Z Shell extension of Bourne shell
  - most featured
  - spelling correction
  - very customizable
- BusyBox – tools and shell in single executable
  - embedded de facto standard

Shell Conventions

- /bin/sh – Bourne (clone) shell - default for root
- /bin/bash – Bourne-Again shell, default for users
- Typically /bin/sh alias (symlink) for /bin/bash
  - May not support same features
- cat /etc/shells – list of installed shells
- Other shells provided for compatibility with existing scripts
- rbash? Restricted shell, e.g. limited, safer

Shell Programming

- Shell programs are scripts containing a series of shell commands
- First line starts with #!
- Tells kernel the script is directly executable
- Follow with the name of shell to execute
  - Use absolute path
- ex. Bourne shell 1st line would be: #! /bin/sh
- ex. Bash shell 1st line would be: #! /bin/bash
- follow 1st line with commands

Shell Programming

- Within the scripts # indicates a comment from that point until the end of the line
- #! a special case if the first characters of file*

```
#!/bin/bash
cd /tmp
mkdir t
```

- Set executable bits on the file with chmod, e.g.:

```
$ chmod +x shell_script
```

* a hashbang or shebang – program loader parses as an interpreter directive (sharp bang)

Built-in Commands

- Bash has several built-in commands
  - Faster than invoking any other program
  - Don’t need to fork a process to execute

- **export** an environmental variable or function
- to all child processes running in current shell
  - export varname=value
  - export –f functionname # exports function
  - example: `$ export country=Canada`
- `env` command lists all environmental variables
- `export -p` command displays exported variables

Built-in Commands

- pwd built-in command to print the current working directory. It basically returns the value of built in variable ${PWD}

```
$pwd
/home/mmaxwell
$echo $PWD
/home/mmaxwell
```

Built-in Commands

- getopts - parse the given command line arguments
- logout - exit a current shell
- umask - sets a file mode creation mask
- unset - set the shell variable to null
  - also used to delete an element of an array
  - to delete complete array.
- printf - similar to C - format print operations
- shopt - set and unset shell options

- *Just a few of the bash built-in commands*

Behavior

- /etc/login.defs Configuration file (shadow)

```
FAIL_DELAY       3
LOGIN_RETRIES    5
ENCRYPT_METHOD   MD5
PASS_MAX_DAYS    100
PASS_MIN_DAYS    0
PASS_MIN_LEN     12
PASS_WARN_AGE    7
ENVIRON_FILE     /etc/environment
```

$ cat /etc/login.defs ugly

$ cat /etc/login.defs | more

Behavior

- /etc/profile Bourne-related System-wide defaults

```
#Set file creation mask to no group/world writable
umask 022
#Tell me when new mail arrives
if [ -x /usr/bin/biff ]; then
  biff y 2> /dev/null
fi
# Set TERM to linux for unknown type or unset variable:
if [ "$TERM" = "" -o "$TERM" = "unknown" ]; then
  TERM=linux
fi
#Make some environment variables global
export PATH TERM
```

/etc/profile

```
# Set the default system $PATH:
PATH="/usr/local/bin:/usr/bin:/bin:/usr/games"
# For root users, ensure that /usr/local/sbin, /usr/sbin, and /sbin are in
# the $PATH   (sshd may not add these by default )
if [ "`id -u`" = "0" ]; then
  echo $PATH | grep /usr/local/sbin 1> /dev/null 2> /dev/null
  if [ ! $? = 0 ]; then
    PATH=/usr/local/sbin:/usr/sbin:/sbin:$PATH
  fi
fi
# For non-root users, add the current directory to the search path:
if [ ! "`id -u`" = "0" ]; then
 PATH="$PATH:."
fi
```

Appends any /etc/profile.d/ scripts

Configurable

- May vary if logon vs interactive start

- .bashrc In your home directory (Examples)

```
alias la="ls -al"
alias diskspace="du -S | sort -n -r |more"
export GREP_OPTIONS='--color=auto'
```

Additional Bash Links

Bash Tutorial
http://www.hypexr.org/bash_tutorial.php

Bash Scripting Tutorials
http://www.panix.com/~elflord/unix/bash-tute.html
http://www.ibm.com/developerworks/aix/library/au-getstartedbash/index.html

Advanced Bash Scripting (Book)
http://www.tldp.org/LDP/abs/html/

Bash Programming
http://www.arachnoid.com/linux/shell_programming.html

Bash Reference Manual
http://www.gnu.org/software/bash/manual/bashref.html

A-Z Index of Bash Command Line
http://ss64.com/bash/

Expanded bash dot files
~~http://dotfiles.org/~mkfs~~

Go look at all of these

Online Bash Tutorial

- Reading/Homework Assignment

IBM. English Sign in (or register)

developerWorks. Technical topics Evaluation software Community Events Search developerWorks

developerWorks > AIX and UNIX > Technical library >

Bash scripting for beginning system administrators

A hands-on approach

Roger Hill (unixman@charter.net), Independent author

Summary: If you're new to Linux® or UNIX® administration and want to get up to speed on bash scripting techniques, or you're a Windows engineer running something like a Cygwin UNIX sub-shell on your system, you need to know the hows, whys, and how-to's for bash shell scripting. Learn everyday usage of bash on a UNIX or Linux system; see how to become a bash power user by chaining bash commands together; and dive into variables, syntax structure, and loops in bash.

Tags for this article: administration, administrators, aix, bash, bash_scripting, beginning, for, os, roger_hill, scripting... **more tags**

Tag this! Update My dW interests (Log in | What's this?)

Date: 12 Oct 2010
Level: Intermediate
PDF: A4 and Letter (106KB | 21 pages) Get Adobe® Reader®
Also available in: Chinese Korean

Activity: 28509 views
Comments: 0 (View | Add comment - Sign in)

Average rating (18 votes)
Rate this article

A UNIX shell is essentially the API between the user, the kernel, and the system hardware. The shell is very important on any UNIX or Linux system and is one of the most vital aspects to learn proper systems administration and security. Typically driven by a CLI, the shell can literally make or break your system. The open source bash shell that this article examines is one of the most powerful, practical, and extensible shells available. In this article, you will learn the basic techniques of bash scripting, its everyday uses, and methods for employing it to create near-bulletproof shell scripts.

History of the bash shell

The Bourne Again Shell (bash) got its start in 1987, when it was written as a GNU project that many Linux distributions quickly adopted. Currently, many different versions of bash are freely available.

Frequently used acronyms

- **API:** Application programming interface
- **CLI:** Command-line interface
- **SNMP:** Simple Network Management Protocol

One of the more positive aspects of bash is its built-in security features. Bash keeps a record of what the user has typed exactly and writes it to a hidden file called *.bash_history* within the user's home directory. So, if you implement bash, you can easily track your systems users much more closely. In fact, for many Linux systems, bash is commonly preinstalled as the default shell environment.

Table of contents

Remember

- shell is a command line interpreter
- A command can be:
  - built-in shell command
  - an executable shell script
  - a source compiled, object code file
- Common shells:
  - bourne, c, bash, ash, korn
- bash prompts for user, root
- hashbang in a script

-EOT-

- Coming Soon!
  - More on Bash