Introduction To Shells

Hour 13

- Objectives
  - Shells: Bourne, Bourne Again, Kom, and C
  - Shell variables and environment variables
  - Starting new shells: sh, csh, ksh, and su
  - Shell configuration or initialization files
  - Terminal settings: the stty command
  - Executing commands with your login shell: . (dot) and source
  - Identifying a running shell

Copyright © 1998-2002 Delroy A. Brinkerhoff. All Rights Reserved.

Shells Are Interactive Programs

The default or login shell is specified in /etc/passwd

- Interpret commands
- Allow users to configure or customize their environment
  - Can be configured from the command line
    - The shell will “forget” this when it is closed or terminated
  - Can read configuration information from configuration files
    - The shell is configured whenever the user logs into the system
- Run programs written in a shell language (see Hour 15)
- Store configuration and other information in variables
  - Shell variables (just used by the shell)
  - Environment variables (used by the shell but also available to all commands or programs)
Four Popular Shells

Shipped with most modern Unix systems

- **Bourne** (/bin/sh; 95,316 bytes)
  - Poor command interpreter / commonly used for programming
  - The traditional language for system configuration scripts
- **C** (/bin/csh; 159,192 bytes)
  - Good command interpreter / rarely used to program
- **Korn** (/bin/ksh; 209,136 bytes)
  - Good command interpreter / good programming features
- **Bourne Again shell** (/bin/bash; 516,392 bytes)
  - Good command interpreter / good programming features
  - Syntax similar to both Korn and C shells (some of the best features of both shells)

Spawning A New Shell

Starting a new shell at the command line

- Your login shell is specified in `/etc/passwd`
  - A few systems support the `chsh` command
  - Have root edit `/etc/passwd` to permanently change your shell
- Spawn or run a different shell with the appropriate command
  - `sh`
  - `csh`
  - `ksh`
  - `bash`
  - Old shell(s) sleep until you close the new shell(s)
- `su [ loginName ]`
  - Switch User / Super User-- spawns a new shell with a new user ID
  - The default login name is root
  - The old shell sleeps until the su-created shell is closed
# Configuration Files

aka initialization or startup files

- **Notes:**
  - Dot files are in the user’s home directory
  - Files are listed in order of execution
  - On *icarus* only, edit configuration files in `$HOME/bin/SunOS5/`
  - Legend: * both login & spawned, ** spawned only, *** logout only

<table>
<thead>
<tr>
<th>csh</th>
<th>ksh</th>
<th>bash</th>
<th>sh</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/etc/.login</code>&lt;br&gt;<code>.cshrc*</code>&lt;br&gt;<code>.login</code>&lt;br&gt;.logout***</td>
<td><code>/etc/profile</code>&lt;br&gt;.profile&lt;br&gt;expand <code>$ENV</code></td>
<td><code>/etc/profile</code>&lt;br&gt;.bash_profile&lt;br&gt;.bash_login&lt;br&gt;.profile&lt;br&gt;expand <code>$BASH_ENV</code>&lt;br&gt;.bashrc**&lt;br&gt;.bash_logout***</td>
<td><code>.profile</code></td>
</tr>
</tbody>
</table>

---

# Shell Variables

Storing and retrieving data

- **C shell**
  - Shell variables are controlled with the `set` command
    - `set` show all variables
    - `set variable="string"` assigns a value to variable

- **Bourne, Bourne Again, and Korn**
  - `variable="string"`
    - quotation marks are needed if string contains spaces or meta-characters
    - note that space on either side of `=` is not allowed

- **Flags are a special case: they are either on or off**
  - Use the `set` command in csh, ksh, and bash
    - e.g.: `set -o ignoreeof`

- **Stored values accessed by variable substitution:**
  - `$variable`
  - `${variable}`
Common Environment Variables

“Global” user information

- Information stored in environment variables can be retrieved and used by other programs (see pp. 289-291)
  - `TERM` - the default terminal type
  - `PATH` - list of paths searched for commands
  - `HOME` - user’s home directory
  - `SHELL` - default or login shell

- Display environment variables
  - `env` - show all environment variables
  - `echo $VAR` - show VAR variable only
  - `echo $TERM`

- It is traditional (not a syntactical requirement) to name environment variables with all upper case letters

Setting Environment Variables

Providing information to other programs

- C shell
  - `<setenv VAR “value”>`
  - `<unsetenv VAR`
  - `<setenv TERM vt100`

- Bourne, Bourne Again, and Korn shells
  - `<VAR=“value”>` - make a shell variable first
  - `<export VAR`
  - `<TERM=vt100`
  - `<export TERM`

- Bourne Again and Korn shells
  - `<export VAR=“value”`
  - `<export TERM=vt100`
An Example With PATH

Locating executables

- **echo** `$PATH`
  ```
  /usr/ucb:/usr/bin:/usr/ccs/bin:/home/csnort/bin:.
  ```
  The dot at the end is the current directory

- **C shell**
  ```
  % setenv PATH $PATH:/tmp
  Variable syntax
  % setenv PATH ${PATH}:/tmp
  % rehash
  ```

- **Bourne, Bourne Again, and Kom shells**
  ```
  $ PATH=$PATH:/tmp
  $ export PATH
  ```

- **Bourne Again and Kom shell**
  ```
  $ export PATH=$PATH:/tmp
  ```

Terminal Settings

The shell interprets the special characters

- **Special characters**
  - `^C` interrupt process
  - `^D` EOF, end of input
  - `^H` or `^?` (Ô/del) erase character
  - `^S` (xoff) stop output
  - `^Q` (xon) resume output
  - `^Z` suspend a process (resume with fg or bg, see Hour 16)

- **Terminal characteristics**
  ```
  stty -a reports current terminal settings
  stty sane sets reasonable terminal settings (may need to use ^J if enter key is not working)
  stty intr ^C
  stty erase ^? or stty erase ^H
  stty kill ^U
  stty eof ^D
  ```
User Defined Configuration Files

Configuring a shell after login

- Share configuration among many users
- Separate large configuration from shell files
- C shell and Bourne Again shell
  `<source [-h] fileName`
  `→ -h` places commands from the file on the history list without executing them
- Bourne, Bourne Again, and Korn shells
  `<. fileName`
  `<Dot (or period) followed by a space and then the name of the file`
- Current shell reads and executes commands in fileName

Identifying Your Shell

The “shell game”

- Login shell
  `<echo $SHELL`
  `<Search the password file; e.g.,`
  `<grep csnort /etc/passwd`
  `→ csnort:x:314:314:Cranston Snort:/home/csnort:/bin/csh`
- Current or spawned shell
  `<source is a C shell only command (used incorrectly); error message reveals the current shell`
  `<source`
  `→ source: Too few arguments`
  `→ ksh: source: not found`
  `→ source: not found`
  `→ bash: source: filename argument required`
  `C shell`
  `Korn shell`
  `Bourne shell`
  `Bourne Again shell`