Chapter 8, Bash

February 16, 2016

Background

- Based on Bourne shell
- Introduced readline library
- Unlike busybox, bash doesn't include work-alike commands for common utilities like tar and rm.

Start-up with bash –login

- If it exists, /etc/profile is first sourced; use strace bash –login to watch; typical contents for this are something like:

```
# /etc/profile: system-wide .profile file for the Bourne sh
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ..

if [ "$PS1" ]; then
  if [ "$BASH" ] && [ "$BASH" != "/bin/sh" ]; then
    # The file bash.bashrc already sets the default PS1.
....
# The default umask is now handled by pam_umask.
# See pam_umask(8) and /etc/login.defs.

if [ -d /etc/profile.d ]; then
  for i in /etc/profile.d/*.sh; do
```

Start-up with bash –login

- After that, typically /etc/bash.bashrc is consulted
- Then /etc/bash_completion is consulted for completion information
- It's hard to believe, but working over the command line completion code is typically the most time-consuming part of the bash start-up.

Start-up with bash –login personalization

- Next comes the personalization code. First ~/.bash_profile is checked
- Then ~/.bash_login
- Then ~/.profile

When you exit

- The file ~/.bash_logout is first consulted
- Then /etc/bash.bash_logout

Start-up of just bash

- First /etc/bash.bashrc is consulted
- Then gobs of completion data are worked through
- Then ~/.bashrc

source, src, or just “dot”

- Most shells support the ability to read more code into the current shell process, much like an “#include” in C
- This is a separate and different ability from executing a child bash process.

source, src, or just “dot”

- The advantage of doing this is that this read-in code can change the current process’s state; a separate child process cannot do so

Scripting

- Bash scripts need two attributes to be standalone:
  - Permission bits must include “x”
  - The first line of the file needs to be formed like “#!/bin/bash”, or perhaps even “#!/usr/bin/env bash”

Short-circuits

- The && and || commands allow for short-circuiting
  - With &&, if the first command fails, then the second one is not executed
  - With ||, if the first command succeeds, then the second is not executed

Subshells

- You can create a subshell with ( ), which particularly useful for backgrounding a statement list

Job control

- fg, bg, and jobs
  - bg sends jobs to the “background”
  - fg brings jobs back to the “foreground”
  - jobs lists all jobs associated with a shell