CST8207 – Linux O/S I

Run Levels, Upstart and Startup Scripts

Topics Covered

- Former runlevels
- Displaying runlevels
- Changing runlevels
- Upstart daemon
- Startup files
- Bash startup

Associated Readings

- Chapter 11 (pages 417 – 424, 430–1, 435)

Linux Runlevels

- Runlevels act as a method to define what processes are started and stopped, and what users are capable of doing by managing each level individually.

- There were 10 runlevels available (***0-9***).
  Red Hat/Fedora Linux uses 7 of them (***0-6***).

Linux Runlevels

- Processes and services are generally associated with a single runlevel at any one time, but can exist in different runlevels concurrently.

- The system can only exist in one runlevel at a time.

  The default runlevel to use at startup is determined by the ***rc*** script set from the information in the ***/etc/inittab*** file

- Runlevels are pre-defined but *can be modified as needed*

Linux Runlevels

- *Default Runlevel definition for most Linux distributions*
  - **Runlevel 0** - Halt
  - **Runlevel 1** - Single user mode (*secure locally*)
  - **Runlevel 2** - Multi-user mode, without NFS
  - **Runlevel 3** - Full multi-user mode (*terminal login*)
  - **Runlevel 4** - undefined
  - **Runlevel 5** - Full multi-user mode (*with an X-based login screen*)
  - **Runlevel 6** - Reboot

Linux Runlevels

- To display current and previous system runlevels:
  - ***runlevel***
    - Print previous and current system runlevels, separated by a signle space. If there is no precious runlevel, “N” will be printed instead

- To change system runlevel
  - ***init***
    - ***init 1***
      - Switch to single mode
    - ***init q*** or ***init Q***
      - Re-examine ***/etc/inittab*** file

Upstart daemon

Upstart Event-Based `init` Daemon

- The traditional System V **`init`** daemon (**`SysVinit`**) does not work well with:
  - Hotplug devices
  - USB hard and flash drives
  - Network-mounted filesystems
- Fedora replaced **`SysVinit`** with the *Upstart **`init`** daemon*.

- Fedora still uses **`xinetd`**
  - Listens for **network** connections
    - Launches a specified server daemon and forwards the data from the connection to the daemon's standard input.

Other traditional init replacements

- Initng (debian and Ubuntu)
- SMF – Service Management Facility (Solaris)
- launchd (MacOS)

- Upstart “plans” to incorporate features from all the above systems
  - Runlevels will not “exist” in Fedora
  - However, runlevels will be maintained for compatibility reasons

Types of init

- **`SysVinit`** uses runlevels and links from the **`/etc/rcX.d`** directories to the init scripts in **`/etc/init.d`** to start and stop system services
- Upstart init uses “events” to start and stop system services
- Since Fedora 9, the transition from **`SysVinit`** to Upstart init

Upstart `init` and events

- Specified programs are run when “something” on the system changes (an event)
  - Usually scripts that start and stop services
  - Similar in layout to that of **`SysVinit`** (init scripts called when a runlevel is changed)
  - However Upstart **`init`** is more flexible
- Example:
  - Upstart calls a script which starts a service when it hears from **`udev`** (utility that manages device naming dynamically) that a printer has been added to the active system.
- Upstart can also start/stop services when the system boots, is shut down or a job changes state.

Jobs (not process control)

- Each file in **`/etc/event.d`** defines a job
  - Usually contains at least an event plus a command

- When an event is triggered, **`init`** executes the command (which could be a script)

Example

- Contents of **/etc/event.d/myjob** file

```
start on myTestEvent
echo "My event has occurred!" > /tmp/myjob.out
```

Or a better implementation…

```
start on myTestEvent
script
    echo "My event has occurred!" > /tmp/myjob.out
    date >> /tmp/myjob.out
end script
```

To execute myjob

- **`script`** and **`end script`** cause **`/bin/sh`** to start (create a shell) and then the commands between **`script`** and **`end script`** will execute

```
initctl emit myTestEvent
```

- The above command will “setup” the event: **`myTestEvent`**

initctl

- **`initctl list XYZ`** will report on the status
- **`initctl emit XYZ`** will trigger a job
- **`initctl start XYZ`** (or simply **`start XYZ`**) will run a job without triggering an event
- **`initctl stop XYZ`** (or simply **`stop XYZ`**) will discontinue the job

Shell Startup Files

What happens when you login?

- After you type in your user name and password, **`/bin/login`** completes execution
  - validating that you are who you claim to be
- **`/bin/login`** then sets up an initial environment (use the command **`env`** to see what was set)
- The bash process then looks into **`/etc/profile`** and executes the commands listed
- It then looks in your home directory for **`.bash_profile`**
- Near the end it executes your personal ENV file: **`.bashrc`**

The Environment – The initialization Files

- /etc/profile
  - system wide initialization file set up by the sys admin
- ~/.bash_profile
  - Potential user defined profile file
  - See page 282 of the textbook for an example
- You will notice that this file immediately loads the contents of another resource file…

~/.bashrc

- If .bash_profile is not found, this file is used
- Contains settings that are used by the bash shell
- Notice that your .bashrc file refers to /etc/bashrc
  - Among other things, this file sets the prompt format with

PROMPT_COMMAND='echo -ne "\033]0;${USER}@${HOSTNAME%%.*}:${PWD/#$HOME/~}"; echo -ne "\007"'

  - Or other prompts depending on where/how you are logged in

~/.profile

- By default this file usually doesn't exist
- You can create this initialization file to create custom environment and terminal settings
- DO NOT use bash specific commands as this file is executed in **`sh`** (the *Bourne Shell*)

~/.bash_logout

- As can be guessed by the name, this file is sourced on logout
- Cleans up any temporary files, truncate the history file, record the time of logout, etc.
- Notice that the .bash_logout file in your home directory does very, very little

Setting bash Options: set and shopt

- Used to customize the shell environment
- Many options available (refer to the man page on bash)

- **`set -X`**, where **`X`** is an option
- In many cases **`set +X`** turns the option off

Some Useful Settings

- **`interactive-comments`**
  - For interactive shells, a leading # is used to comment out any remaining text
- **`noclobber`** (**`-C`**)
  - Protects files from being overwritten when redirection is used (VERY USEFUL!)
- **`notify`** (**`-b`**)
  - Notifies user when a background job completes
- **`nounset`** (**`-u`**)
  - Displays an error when expanding a variable that has not been set
- **`verbose`** (**`-v`**)
  - Turns on debugging (such as it is)

Shopt (bash versions 2+)

- The shopt builtin is an alternate to set and adds many more options
- As usual, refer to the man page on bash

The Primary Prompt

- $ is the default primary prompt
- As seen earlier, we can over-ride this and use our own
- The PS1 environmental variable can be set to change the prompt at will using various escape sequences (listed on page 307 – 309 of the textbook)

PS1 codes

- \$ # if the user is root; otherwise $
- \w Pathname of the working directory
- \W Basename of the working directory
- \! Current event (history) number
- \d Date in Weekday Month Date format
- \h machine hostname, without domain
- \H full machine hostname
- \u username of the current user
- \@ Current time of day in 12hr AM/PM
- \T Current time of day in 12hr HH:MM:SS
- \A Current time of day in 24h HH:MM
- \t Current time of day in 24h HH:MM:SS

A Few Prompt Examples

- PS1=[\u@\h \W]\$ gives
  - [user@wt127-31 etc]$ when “user” is logged into machine # 31 and is in the /etc directory
- PS1='[\u on \h at \t, \!]\$‘ gives
  - [user on wt127-31 at 12:21:51, 1024]$

The Search Path

- The env variable PATH is used to determine where commands could exist
- echo $PATH lists the order and where to look
  - /usr/local/bin:/bin:/usr/bin

Customizing The PATH

- To add the current directory to the search path, we need to add it to the search path
- PATH=$PATH:.
- Unfortunately, this must be done each time we log in or create a new terminal
- Where can we set this to make it permanent?

Environmental Variables

- Effectively global variables accessible from within processes and child processes
- Can be created with the **`export`** command (or **`delare -x`**)
- By convention they are CAPITALIZED
- Many, many variables available and listed in the bash man page (as well as many other resources)