Runlevels intro

Will Trillich

will@dontUthink.com

What do you want here as an abstract, Will?

1. Assumptions

I’m writing this in January of 2001 ("Open the pod bay doors, Hal") as it applies to my Debian "potato" release (that is, the uname -a command tells me I’m running Linux kernel 2.2.17). Your mileage may vary -- chances are SOMETHING in your system is different from mine, right?

Debian "Potato" Release Kernel 2.2.17 Other Linux distributions MAY find this information useful as well, but it’s tailored for the Debian "Potato" crowd.

By the way -- for you Mac and Windows folks out there -- my Debian system, with exim and mutt for email, and web and ftp servers, has been up without any hiccups for 108 days as of today. (My Mac has stretches when it behaves delightfully for weeks, and then it freezes up in spurts, five times a day for the next ten days... Windows folks know the feeling, too.)

That "fits-and-starts" syndrome just doesn’t happen with Linux.

2. Introduction

2.1. Credits

Tom Huckstep converted the HTML version to DocBook.

2.2. This probably needs moving out of the intro

Sure, you can ls your directory, and grep a file here and there – and you create the process at your shell prompt... And you created the shell prompt when you first logged in...

Ever wonder: So what creates the first process? (If not, you should.) What starts all these processes? Where exactly IS the God of the machine?

If you haven’t tinkered with ps yet, now’s the time.

```
$ ps axf
```

And to learn about **ps**, try

```
$ man ps
```

What **ps** does is show you the current processes that exist – including any **ls** or **grep** or **find** or **emacs** – many will be idle, as they wait for some signal (a press at the keyboard, a network packet, a signal from the printer). Others will be actively using processor brainpower.

Using **ps axf** (instead of just plain **ps** without any arguments) makes it display ALL processes (not just those owned by you, or connected to certain terminals, but ALL processes background or otherwise) in a family-tree format. Here you can see which process spawned which others, for several 'generations'. Here's a trimmed-down cross-section of my system at the moment:

```
$ ps afx
 PID TTY      STAT   TIME COMMAND
 5790 ?        S      1:41 /usr/sbin/xinetd
25445 ?        S      0:00  \_ in.telnetd: wdt
25446 pts/0    S      0:00      \_ -tcsh
25878 pts/0    T      0:00          \_ vi runlevels-intro.wml
26045 pts/0    R      0:00          \_ ps afx
```

I edit out lots of extraneous lines to reduce distractions; you'll always see a heck of a lot more than just five lines, in your **ps axf** output. (By the way -- don't tell anyone I use **telnet** [a.k.a. Satan's Favorite Evil Bad Security Breach from Hell] even though I've got it locked down via **xinetd** to interact only with my local intranet. Do as I say, not as I do: use **sshd** instead!)

The above listing confirmed our initial suspicions about logging in creating the shell, which creates all the other command processes we execute: you can see that **xinetd** spawned the **in.telnetd** process, which in turn is the parent process of my **tcsh** login shell which has two child processes. I am editing this very file (the **vi runlevels-intro.wml** command) which I suspended (with **Ctrl**-**Z** so I could get back to the **tcsh** command shell) in order to run the **ps afx** command – so I could cut & paste it into this document.

Below, I run the **ps** program again, but ask for a LONG listing via **ps afxl**:

```
$ ps afxl
  F   UID   PID  PPID STAT TTY    TIME COMMAND
100     0     1     0 S    ?      0:17 init [2]
140     0  5790     1 S    ?      1:41 /usr/sbin/xinetd
100   101 25445  5790 S    ?      0:00  \_ in.telnetd: wdt
100  1000 25446 25445 S    pts/0  0:00      \_ -tcsh
000  1000 25878 25446 T    pts/0  0:01          \_ vi runlevels.wml
000  1000 26072 25446 R    pts/0  0:00          \_ ps axfl
```

(Again, I've whittled down on the rows, and this time, the columns, too. Your display will have lots more info than this, guaranteed.) Notice that the process-ID (PID) of **tcsh**, which is 25446, also shows up as the parent process ID (PPID) for **vi** and **ps** ? That's the "family tree" info that's needed in order to display the forest/family-tree style indentation; every running process knows its parent process.

Now -- notice how the PPID (parent) of **xinetd** is PID #1? That's **init**, which turns out to be the granddaddy of them all. Quite literally! (**ps** doesn't display **init** in the family tree because it's kinda "understood" that everything owes its existence to **init**.)

Think of init as kinda like Adam in his Garden of Eden, without any worries about sufficient diversity in the gene pool. He just clones himself, and then the clone takes on the function of the requested command process, which may in turn clone itself for the next job, and so on.

3. Init is the granddaddy of them all

All processes are spawned by **init**, or by processes whose ancestor is **init**. Period!

Great. So how the heck does **init** know what to do?

When **init** starts up at boot time it looks in `/etc/inittab` for instructions. In my `/etc/inittab`, for example, the first non-comment is

```
id:2:initdefault:
```

which tells **init** that until another runlevel is specified, wake up using 'runlevel 2' by default.

But even before the runlevel-specific stuff is executed, the 'system-wide gotta-have' stuff is specified in `/etc/inittab` via

```
si::sysinit:/etc/init.d/rcS
```

which runs all the initial stuff, like loading kernel modules, mounting partitions, hardware clock settings, yada yada... What the `/etc/init.d/rcS` script does (go have a look at yours, I'll wait) is run all the scripts in `/etc/rcS.d/S*` to get all your basic system features up and running before delving into the items specific to each runlevel. For example, you might need network cards and SCSI drivers up-and-at-em – but you don't need any SQL engines or webservers until you're entering runlevel (pick one, for example) four.

Whenever you boot into single-user mode (also called 'runlevel S') **init** runs only **rcS** stuff and doesn't continue any further – enabling you to log in as root and twiddle with your settings, before starting a services-oriented runlevel. (By services-oriented, I mean something like "on runlevel A we've got **X** and **apache** and **nfs** available; on runlevel B we have only **sshd** with logins restricted to group XYZ only; on runlevel C we have **nfs** and **ftp** ...")

3.1. But how about the feature-sets for a "user runlevel"?

For the runlevel-specific stuff (and this is what determines the difference between runlevel 3 and 5 and 2 and ...) further down in `/etc/inittab` there's

```
l2:2:wait:/etc/init.d/rc 2
```

which says that for runlevel 2, to run the `/etc/init.d/rc` script with argument 2. That's the script that you'll find that does most of the work. Go check it out.

Anyhow, what the `/etc/init.d/rc` script does is look in

```
/etc/rc[runlevel-digit].d/*
/etc/rc2.d/*
```

for scripts to run. A very nice, modular structure. And all those `/etc/rc*.d/*` items are just symlinks to the actual scripts in `/etc/init.d/` anyhow.

3.2. What makes init start up a login sequence?

Even further down in `/etc/inittab` is the procedure for establishing a live tty connection, which enables you to log in in the first place:

```
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6
```

According to this, for runlevels 4 and 5, only `tty1` will be active; for 2 and 3 `tty[1-6]` will all be active. (It's actually more sensible to say that tty1 is gonna be active for runlevels 2-5, and `tty[2-6]` will be active only for runlevels 2-3.) And when the connection goes down (either you log out in a nice, neighborly fashion, or you get rudely disconnected by a power failure or because your neice chewed through your modem cable), init knows to 'respawn' the process for the next victim. Cool, eh?

3.3. What runlevel am I in?

How do you find out which runlevel you're in right now? This one's easy. Try this:

```
$ /etc/runlevel
S 2
```

It displays both the previous runlevel (which may be S after a successful system startup, or perhaps N, signifying that there was no previous runlevel) and the current runlevel. (My system is at runlevel 2 after last being at the single-user [S] runlevel.) Nothing to it.

3.4. Changing to a different runlevel

This is not something to do lightly, especially if you have several users on your system. They'll track you down eventually (and have been known to use weapons) and you'll regret your careless act unless you have a good reason...

If it's really a good idea to do so, here's the recommended way to change from whatever runlevel you're using, to another runlevel:

```
# telinit 4
# telinit S
# telinit 2
```

Very simple. Just **telinit** (*as root*) which runlevel to switch to, and you're off!

3.5. How do the runlevels differ?

Sorry, I can't answer that... But you can.

Remember the `/etc/inittab` section that showed what to do on each runlevel?

```
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
```

They all run the same script -- /etc/init.d/rc ! The very only single difference is, which argument is sent to that script.

So we hafta check into that script to see what the argument does.

If you look at the /etc/init.d/rc script, you'll find portions that look something like this:

```
# Is there an rc directory for this new runlevel?
if [ -d /etc/rc$runlevel.d ]
...
for i in /etc/rc$runlevel.d/K[0-9][0-9]*
...
for i in /etc/rc$runlevel.d/S*
```

Can you see what that does? If you enter runlevel 3 (perhaps via **telinit 3**) it'll try running scripts from `/etc/rc3.d/*`. Mystery solved.

When entering a runlevel, you may need to "turn off" features that might have been turned on by another runlevel. Then, you turn on the features for the new runlevel.

So first, `/etc/init.d/rc` will run all the "kill" scripts (any script whose name starts with "K") in the new run level with a "stop" argument. For example:

```
/etc/rc3.d/K20postgresql stop
```

Which stops the pgsql server for runlevel 3

After it's all done running available "kill" scripts it then runs the "start" scripts (which have names staring with "S") in much the same way:

```
/etc/rc3.d/S60sshd start
```

Which launches the secure shell daemon for runlevel 3.

So now you should see why YOU can answer the question better than I can: only YOU can determine the difference between your runlevels ... by looking at your `/etc/inittab` file and by checking out the `/etc/rc*.d/*` scripts!

And note that the scripts will be run "in sequential order" meaning that `S10*` would run before `S60*` and so forth. This is how you can ensure that the load order works properly. For example, you may need to establish a network connection with a fileserver *before* launching a remote-log daemon.

BUT! There are certain preset runlevels that have important meanings:

- **telinit 0** = SHUTDOWN!
- **telinit 1** = Single user (root only) mode
- **telinit 0** = REBOOTS your system!

4. Changing what a runlevel does

There are Debian ways of munging the directory of scripts that the `/etc/init.d/rc` script calls upon.

4.1. Create a script to start or stop a feature

In order to affect a runlevel, you need to create a script that does some work.

```
# cd /etc/init.d
# cp skeleton my_sample_script
```

That skeleton is a nice way to start you off on the right foot. Once you get your `my_sample_script` working, you can add it to various runlevels.

4.2. Determine where it belongs in the START or STOP sequence

Even when your script works properly, you'll need to know where it belongs in the startup sequence for example, to check out runlevel 4:

```
# cd /etc/rc4.d
$ ls -F
S10sysklogd@  S20gpm@       S20postgresql@  S50wu-ftpd@  S99rmnologin@
S12kerneld@   S20inetd@     S20ssh@         S60sshd@
S19bind@      S20logoutd@   S20xinetd@      S89atd@
S20cipe@      S20makedev@   S22ntpdate@     S89cron@
S20exim@      S20mysql@     S23ntp@         S91apache@
```

This sample, taken from my system, shows there are zero KILL scripts, and 21 START scripts. If you wanted your new script to be STARTed after sshd (which is number 60) and before atd (number 89) then using any number near S70 for startup would do the trick.

Let's say you also want to run the KILL portion (maybe to refresh or disable something, it's up to you) when entering runlevel 4, as well. In my example above, I have NO kill scripts, so there's no established sequence to fit into so any number will do, but let's use 50.

4.3. Install the links that call the script

Here's how you set up your script to START as sequence #70 and STOP as sequence #50 for runlevel 4:

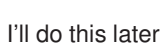
```
# update-rc.d my_sample_script start 70 4 . stop 50 4 .
```

That'll create links in `/etc/rc4.d/` called `K50my_sample_script` and `S70my_sample_script` that point to your original script. The actual script itself is in `/etc/init.d/my_sample_script`.

You could also add your start/stop script to several runlevels at once. For example, add it to runlevels 2 through 5 like this:

```
# update-rc.d my_sample_script start 70 2 3 4 5 . stop 50 2 3 4 5 .
```

You'll wind up with this:



Easy as pie!

4.4. Removing a feature from a runlevel

When you decide to REMOVE your script (or any other service) from a runlevel, do like so:

```
# cd /etc/init.d
# mv my_sample_script my_sample_script.DISABLED
# update-rc.d my_sample_script remove
```

If you don't remove (or rename, if you're anal like I am and deplore actually deleting anything, ever) the actual script first, you'll need to supply other arguments to update-rc.d to get rid of the links.

Of course you can always tangle things up manually if you like, by editing /etc/init.d/* scripts and munging links yourself... but that way, lies madness.

5. Need to know more?

Don't stop here! Learn more by trying these:

```
$ man init
$ man inittab
$ man ps
$ man update-rc.d
http://packages.debian.org/procps
http://www.eGroups.com/files/newbieDoc
```