



Managing Initscripts with Red Hat's chkconfig

SysAdmin (/tag/sysadmin)

by Jimmy Ball on April 1, 2001

I love discovering new UNIX commands, especially those with a system administration flavor. When I learned Red Hat distributed the **chkconfig** utility, it brought back fond memories of chkconfig under IRIX, a UNIX variant from Silicon Graphics, Inc. IRIX's chkconfig was designed to enable/disable services for automatic launch during system initialization without editing, renaming or moving initscripts in **/etc**.

Similarly, Red Hat designed chkconfig to help manage services launched during system initialization. But, after perusing the man page and doing some tests, I soon found that Red Hat extended chkconfig with finer control of system startup/shutdown tasks by managing the symbolic links to initscripts. It's a real time-saver!

Startup Basics

When your Linux box boots, the first process that shows up is **init**. If you haven't seen init before, take a moment to type **ps -ef | grep init** to see the PID of init. In short, the init performs tasks that are outlined in **/etc/inittab**.

Some tasks outlined in **/etc/inittab** will be launched soon after init, while others are simply set up. For example, the default Red Hat **/etc/inittab** sets up a trap for the key sequence Ctrl-Alt-Delete. When these keys are simultaneously pressed at a console prompt (not xdm), the shutdown command is performed. At boot time, init sets up this feature based on configuration options in **/etc/inittab**, but execution is postponed until the key sequence occurs.

The format of `inittab` allows for comment lines beginning with a “#” symbol while normal entries are “:” delimited. They follow the pattern **id:runlevel:action:process** where `id` represents a user-defined and unique identifier, `runlevel` can be a combination of the numbers 0-6 or just left blank, `action` comes from a keyword that describes how `init` should treat the process, and `process` is the command to execute.

Descriptions of various keywords for the action field can typically be found in the man pages for `inittab`. Common keywords across most, if not all, UNIX platforms include:

- `initdefault`—defines the runlevel to enter once the system has booted.
- `wait`—a process that will be executed once (when the runlevel is entered). The `init` process will wait for this process to terminate.
- `boot`—defines a process that is executed at boot time.
- `bootwait`—similar to `boot` but `init` waits for the process to terminate before moving on.
- `sysinit`—defines a process that is executed at boot time before any `boot` or `bootwait` `inittab` entries.

The runlevel field designates system state. For example, a runlevel of 0 corresponds to a halted system while a runlevel of 6 corresponds to a system reboot. Unfortunately, all Linux distributions do not follow the same definition for runlevels. Under Red Hat, the following defaults are supported:

0. System halt
1. Single-user mode
2. Multiuser, without NFS
3. Complete multiuser mode
4. User defined
5. X11 (XDM login)
6. Reboot

For each runlevel, there is a corresponding directory in `/etc/rc.d`. For a runlevel of 5, the directory `/etc/rc.d/rc5.d` exists and contains files related to tasks that need to be performed when booting into that runlevel. Under Red Hat, these files are typically symbolic links to shell scripts found in `/etc/rc.d/init.d`.

Let's put this all together with a simple example. Below are two sample lines from our `inittab` file:

```
id:3:initdefault:
13:3:wait:/etc/rc.d/rc 3
```

Here is a typical scenario of what happens under Red Hat. Once init is started, it reads `/etc/inittab` (see above). From the first line, we know that init is going to end up at a runlevel of 3 after the system boots. Once we reach that runlevel, the second line tells init to run the script `/etc/rc.d/rc` three and waits for it to terminate before proceeding.

The script `rc` in `/etc/rc.d` receives 3 as an argument. This 3 corresponds to a runlevel of 3. As a result, the `rc` script executes all the scripts in the `/etc/rc.d/rc3.d` directory. It first executes all the scripts that begin with the letter K (meaning “kill” the process or service) with an argument of “stop”. Next, it runs all the scripts that begin with the letter S with an argument of “start” to start the process or service. As one final note, the order of K and S script execution is based on sort order; the script named `S90mysql` would execute before the script named `S95httpd`.

It turns out the scripts in `/etc/rc.d/rc3.d` are actually symbolic links to scripts residing in `/etc/rc.d/init.d`. While the UNIX administrator can place scripts in `rc3.d`, the common practice under Red Hat is to first place all scripts in `init.d`, then create logical links to the `rc*.d` directories. It doesn't take long to figure out the creation and maintenance of these scripts and symbolic links could be quite the chore. That's precisely where `chkconfig` steps in! The Red Hat `chkconfig` utility is specifically designed to manage the symbolic links in `/etc/rc.d/rc[0-6].d`.

Viewing `chkconfig` Entries

The `chkconfig` binary resides in `/sbin` with default permissions that allow any user to execute it, although the user without root privileges can only view the current `chkconfig` configuration. So type

```
[root]# chkconfig --list | grep on
```

A partial listing of the output would look similar to the following:

```
amd 0:off 1:off 2:off 3:off 4:on 5:on 6:off
apmd 0:off 1:off 2:on 3:off 4:on 5:off 6:off
arpwatch 0:off 1:off 2:off 3:off 4:off 5:off 6:off
atd 0:off 1:off 2:off 3:on 4:on 5:on 6:off
autofs 0:off 1:off 2:off 3:off 4:off 5:off 6:off
named 0:off 1:off 2:off 3:off 4:off 5:off 6:off
bootparamd 0:off 1:off 2:off 3:off 4:off 5:off 6:off
keytable 0:off 1:off 2:on 3:on 4:on 5:on 6:off
crond 0:off 1:off 2:on 3:on 4:on 5:on 6:off
syslog 0:off 1:off 2:on 3:on 4:on 5:on 6:off
netfs 0:off 1:off 2:off 3:on 4:on 5:on 6:off
network 0:off 1:off 2:on 3:on 4:on 5:on 6:off
```

On each line of output, the first field represents the name of an initscript in `/etc/rc.d/init.d`. The remaining fields correspond to the runlevels 0-6 along with the status of the script when entering that runlevel. For example, **crond** would be launched when entering the runlevels 2, 3, 4 and 5 and stopped when entering the runlevels 0, 1 and 6. We can confirm that these settings are true with the **find** command to search for all files in **/etc/rc.d** that end with **crond**:

```
[root]# find /etc/rc.d -name '*crond' -print
/etc/rc.d/init.d/crond
/etc/rc.d/rc0.d/K60crond
/etc/rc.d/rc1.d/K60crond
/etc/rc.d/rc2.d/S40crond
/etc/rc.d/rc3.d/S40crond
/etc/rc.d/rc4.d/S40crond
/etc/rc.d/rc5.d/S40crond
/etc/rc.d/rc6.d/K60crond
```

Notice for each “off” section reported by `chkconfig` (0, 1, 6), a kill script is in place and for each “on” section reported (2, 3, 4, 5), a start script exists. Next, execute a different `find` command to determine the type of each file found:

```
[root]# find /etc/rc.d -name '*crond' -exec file {} \;  
/etc/rc.d/init.d/crond: Bourne shell script text  
/etc/rc.d/rc0.d/K60crond: symbolic link to  
    ../init.d/crond  
/etc/rc.d/rc1.d/K60crond: symbolic link to  
    ../init.d/crond  
/etc/rc.d/rc2.d/S40crond: symbolic link to  
    ../init.d/crond  
/etc/rc.d/rc3.d/S40crond: symbolic link to  
    ../init.d/crond  
/etc/rc.d/rc4.d/S40crond: symbolic link to  
    ../init.d/crond  
/etc/rc.d/rc5.d/S40crond: symbolic link to  
    ../init.d/crond  
/etc/rc.d/rc6.d/K60crond: symbolic link to  
    ../init.d/crond
```

This reveals that crond found inside init.d is a shell script and all remaining files found are symbolic links to the crond script.

Modifying chkconfig Entries

Modifying a chkconfig entry is almost as easy as listing the existing configuration. The form is:

```
chkconfig [--level <levels>] <name> <on|off|reset>
```

For example, if we decide to disable crond for runlevel 2, the **chkconfig --level 2 crond off** command (executed by root) would turn off crond for the runlevel of 2. Running **chkconfig --list** will confirm that crond's configuration has been modified. Further, the find command below reveals that a kill script has replaced the start script in the rc2.d directory:

```
[root]# find /etc/rc.d -name '*crond' -print
/etc/rc.d/init.d/crond
/etc/rc.d/rc0.d/K60crond
/etc/rc.d/rc1.d/K60crond
/etc/rc.d/rc2.d/K60crond
/etc/rc.d/rc3.d/S40crond
/etc/rc.d/rc4.d/S40crond
/etc/rc.d/rc5.d/S40crond
/etc/rc.d/rc6.d/K60crond
```

Keep in mind that chkconfig does not automatically disable or enable a service immediately. It simply changes the symbolic links. The superuser could disable the crond service immediately with the command `/etc/rc.d/init.d/crond stop`. Finally, you can enable/disable a command for multiple runlevels with one chkconfig command. For example entering

```
chkconfig --levels 2345 crond on
```

would set up crond to be started for runlevels 2, 3, 4 and 5.

Removing an Entry

At times, removing a service altogether may be in order. Take sendmail, for example. On client machines where incoming mail for local accounts is not required, running sendmail as a `dæmon` may not be necessary. In this case, I find disabling sendmail desirable since it reduces potential security risks. To remove sendmail from chkconfig, type

```
chkconfig --del sendmail
```

Below, our find command shows no symbolic links in place, while the initscript for sendmail remains:

```
[root]# find /etc/rc.d -name '*sendmail' -print
/etc/rc.d/init.d/sendmail
```

This is perfect in my opinion. The script is left in case sendmail needs to be re-established as a service, but all symbolic links are gone. While we could have disabled sendmail for all runlevels, this would have placed kill scripts in each of the `rc*.d` subdirectories, an unnecessary task since sendmail was never launched during the initialization phase. However, I have seen situations when the system administrator

would manually start a service on certain occasions. Having the kill scripts in place for that service ensures a cleanly killed service. So, you make the call.

Adding a chkconfig Entry

So far, so good. We've seen how to view, modify and delete services using chkconfig. It's time to add a new service. Take the script named **oracle** (see Listing 1).

[Listing 1. Oracle Script \(/files/linuxjournal.com/linuxjournal/articles/044/4445/4445l1.html\)](/files/linuxjournal.com/linuxjournal/articles/044/4445/4445l1.html)

Using this script, Oracle 8 can be started with the “start” argument and terminated with the “stop” argument. This meets the minimum requirements of an initscript that can be used in conjunction with the launch script /etc/rc.d/rc.

Place the script in /etc/rc.d/init.d and run (as root)

```
chmod +x /etc/rc.d/init.d/oracle
```

to make the script executable. If you are concerned about normal users seeing the script, you could try more restrictive file permissions, as long as the script is executable by root as a standalone script.

Notice the two comments lines in the script:

```
#chkconfig: 2345 80 05  
#description: Oracle 8 Server
```

These lines are needed by chkconfig to determine how to establish the initial runlevels to add the service as well as set the priority for the start-and-stop script execution order. These lines denote the script will start Oracle 8 server for the runlevels 2, 3, 4 and 5. In addition, the start priority will be set to 80 while the stop priority will be 05.

Now that the script is in place with the appropriate execute permissions and the required chkconfig comments are in place, we can add the initscript to the chkconfig configuration by typing, as root, **chkconfig --add oracle**.

Using chkconfig's query feature, we can verify our addition:

```
[root]# chkconfig --list | grep oracle  
oracle          0:off    1:off    2:on     3:on     4:on     5:on     6:off
```

Also, we can type our standard find command to see how chkconfig set up the symbolic links:

```
[root]# find /etc/rc.d -name '*oracle' -print
/etc/rc.d/init.d/oracle
/etc/rc.d/rc0.d/K05oracle
/etc/rc.d/rc1.d/K05oracle
/etc/rc.d/rc2.d/S80oracle
/etc/rc.d/rc3.d/S80oracle
/etc/rc.d/rc4.d/S80oracle
/etc/rc.d/rc5.d/S80oracle
/etc/rc.d/rc6.d/K05oracle
```

As requested, the names of the kill links contain the priority 05 while the start links contain 80. If we need to adjust the priorities, (e.g., our stop priority needs to be 03), simply modify the chkconfig comment lines in the initscript for oracle and run the **reset** command, as shown below. The resulting symbolic links will be renamed accordingly:

```
[root]# chkconfig oracle reset
[root]# find /etc/rc.d -name '*oracle' -print
/etc/rc.d/init.d/oracle
/etc/rc.d/rc0.d/K03oracle
/etc/rc.d/rc1.d/K03oracle
/etc/rc.d/rc2.d/S80oracle
/etc/rc.d/rc3.d/S80oracle
/etc/rc.d/rc4.d/S80oracle
/etc/rc.d/rc5.d/S80oracle
/etc/rc.d/rc6.d/K03oracle
```

Enhancements in Red Hat 7

As many of you already know, **inetd** was replaced by **xinetd** in Red Hat 7. In addition, chkconfig functionality has been extended to manage some of the functionality of xinetd's Internet services. Sample output is shown below:

```
[root]# chkconfig --list
...
xinetd based services:
  finger:  on
  linuxconf-web:  off
  rexec:  off
  rlogin:  off
  rsh:  off
  ntalk:  off
  talk:  off
  telnet:  on
  tftp:  off
  wu-ftpd:  on
```

To disable a xinetd feature, perhaps finger, you could type **[root]# chkconfig finger off**.

Pretty neat, huh? However, there is one “gotcha”. When the configuration is changed, the xinetd is signaled automatically to reload the new configuration with the command `/etc/init.d/xinetd reload`, that is executed by chkconfig. This script performs a kill with the SIGUSR2 signal which instructs xinetd to perform a hard reconfiguration.

What does that mean? Well, when I tested it, the active sessions of services offered through xinetd (i.e., Telnet, FTP, etc.) were immediately terminated. That might not be a problem for you, assuming you can plan the best time to disable/enable xinetd services on your system. As an alternative, you can modify the `/etc/init.d/xinetd` script so that the reload option sends a SIGUSR1 signal, which is a soft reconfiguration. This will restart the services without terminating existing connections.

Adding xinetd services for chkconfig management is as simple as adding an xinetd service file into the `/etc/xinetd.d` directory. The chkconfig utility will automatically pick it up and make it available for management through the chkconfig utility. Neat!

Conclusion

Hopefully, you've seen the benefits of Red Hat's chkconfig utility for managing initscripts. While its functionality seems simple, the timesaving benefits makes chkconfig an administrator's command worth committing to memory.

 Managing Initscripts with Red Hat's chkconfig

Jimmy Ball is an instructor with Batky-Howell, Inc. where he teaches UNIX, Perl and Java courses. He can be reached by e-mail at jb@batky-howell.com.

Load Disqus comments (https://www.linuxjournal.com/article/4445#disqus_thread)

You May Like



Simplifying Linux System Administration with Webmin (</content/simplifying-linux-system-administration-webmin>)

George Whittaker (</users/george-whittaker>)

(</content/simplifying-linux-system-administration-webmin>)

Connect With Us



(<https://youtube.com/linuxjournalonline>)



(<https://www.facebook.com/linuxjournal/>)



(<https://twitter.com/linuxjournal>)

Linux Journal, representing 25+ years of publication, is the original magazine of the global Open Source community.

© 2025 Slashdot Media, LLC. All rights reserved.

[PRIVACY POLICY \(https://slashdotmedia.com/privacy-statement/\)](https://slashdotmedia.com/privacy-statement/) |

[TERMS OF SERVICE \(https://slashdotmedia.com/terms-of-use/\)](https://slashdotmedia.com/terms-of-use/) |

[ADVERTISE \(/sponsors\)](/sponsors)

MASTHEAD
(</CONTENT/MASTHEAD>)

AUTHORS (</AUTHOR>)

CONTACT US
(</FORM/CONTACT>)

RSS FEEDS ([/RSS_FEEDS](/RSS_FEEDS))

ABOUT US (</ABOUTUS>)