# Slackware Linux 101



## A look at what happens when you boot your Linux box

Joe "Zonker" Brockmeier (jbrockmeier@earthlink.net)
Senior Editor, User Friendly Media
March 2001

Joe Brockmeier examines the Slackware Linux init sequence. He talks about how the system initializes services, what the various runlevels are and how to add or remove services from the default install to customize your system.



Contents:



Related dW content:

Setting up a Local Area Network

Linux users are increasingly becoming power-users, which means they want to configure their system to do exactly what they want. But these days Linux distributions usually come with autoconfigured devices and start services, like Sendmail or Apache. What distributors don't take into account is that running services like Apache with their default settings intact -- unknown to the user -- is open season for crackers and script kiddies. And it eats up system resources that could be put to better use -- like more processor time for Quake or your favorite compiler. Since lack of control is a Bad Thing, let's look at what happens when a Linux system boots during the init process, at the various runlevels this involves, and how to customize your system or switch between runlevels while your system is running.

Our examples use the Slackware Linux distribution on the x86 platform (see Resources later in this article). Most of the information carries over to other Linux distributions, but there may be some discrepancies in the specifics. In particular, Slackware's init structure is more akin to the BSD UNIX structure than the System V structure, though with the latest distribution of Slackware there are some concessions for programs that want to add services to startup but expect a System V directory structure. (See the sidebar, "The difference between BSD and System V init scripts".)

### The parent of all processes

What actually happens when a Linux box boots up? After your computer's BIOS has done its thing, the system reads the first bit of your hard drive (or floppy, or CD-ROM, or Zip drive...Linux is very flexible) and encounters the bootloader. Usually this is the LInux LOader, better known as LILO, though GRUB and other bootloaders are becoming popular too.

LILO then loads the Linux kernel into memory and it starts to work its magic. The Linux kernel initializes devices like SCSI cards and other hardware devices that are built into the kernel. Then the kernel runs init, which is the first process running on your system besides the kernel. If you do a ps ax | grep 1 you'll see that init has the first process id (PID).

After init is loaded, it reads the inittab to see what to do next. The inittab tells init what runlevel to go into and where the configuration file for that runlevel can be found.

**Runlevels**
A runlevel is defined by all the services available on the system at a given time (basically, the mode of operation). Linux can be in several modes of operation: single-user mode, single-user with networking, multi-user mode, multi-user starting in X, etc. This section will explain the concept of a runlevel, what runlevels are available in Slackware, and what they're called.

Runlevels are designated by number or letter. Unfortunately, not all Linux distributions agree on what the various runlevels should be called. In some distributions runlevel 3 is multi-user with an X login. Others, like Slackware, designate runlevel 3 to be multiuser with console login.

All Linux distributions, at least that I'm aware of, agree that runlevel 0 is "halt," runlevel 1 or "S" is single user mode (more about that in a minute), and runlevel 6 reboots the system. With Slackware it breaks down like this:

- Runlevel 0 = System Halt
- Runlevel 1 = Single user mode, mostly used for maintenance
- Runlevel 2 = Unused
- Runlevel 3 = Multiuser with console login
- Runlevel 4 = Multiuser with an X11 session manager (XDM, GDM, KDM)
- Runlevel 5 = Unused
- Runlevel 6 = Reboot
- Runlevel S or s = Single user mode

There are also undocumented runlevels 7 through 9, which are theoretically available for custom runlevels if they're needed. But I haven't ever personally tried to set one up.

**Runlevel configuration**
If you're not using Slackware Linux, your configuration files are going to be arranged quite differently than the structure I'm about to talk about. Other than the inittab file, all of Slackware's start-up configuration files reside in the /etc/rc.d/ directory.

There are five runlevel rc.* scripts in the directory by default, six if you count the symlink from rc.0 to rc.6.

The runlevel init scripts are:

- rc.0 = The rc.0 file is a symlink to rc.6
- rc.M = The init script for multi-user runlevels 2, 3, and 5
- rc.K = The "administrative" runlevel, single user mode
- rc.S = The system initialization script
- rc.4 = The init script for runlevel 4, which is the runlevel that automatically boots into the X Session Manager of choice
- rc.6 = The script executed by init when rebooting or halting the system

The remainder of the rc.* files in the directory start up system services like networking, kernel modules, PCMCIA, Samba, Apache, Netatalk, and GPM. If you want to make a service like Apache completely unavailable for any runlevel, use chmod to change the permissions on the file from executable to non-executable. You could probably achieve the same effect by removing the file, but I don't recommend that. You may find you want to re-enable a service at a later date and don't know how.

**The difference between BSD and System V init scripts**
It's easy enough to say that Slackware Linux uses BSD-ish init scripts, but what does that actually mean?

Linux, and UNIX, systems that emulate a BSD-ish init style have one directory, /etc/rc.d/, that contains an init script for each runlevel. So if you want to look at or modify the init script for runlevel 4, you want the file /etc/rc.d/rc.4.

On the other hand, systems that base their init scripts on System V have separate directories for each runlevel. So if you want to edit the runlevel 4 init scripts, you look for the /etc/rc4.d/ directory and try to find the script inside that directory that corresponds to the service you want to modify.

The difference hails back to design differences in Berkeley Software Distribution (BSD) UNIX and AT&T System V UNIX, both of which made their way into various commercial versions of UNIX. This, along with Vi vs. Emacs, is one of the holy wars of computing.

The rc.inet1 script is responsible for starting the base networking services like setting the hostname (IP and DHCP). The rc.inet2 script is responsible for starting all other INET services like NFS, packet forwarding, ssh server, and other networking daemons.

> Because many of the Linux distributions favor the System V init layout, Slackware now comes with the directories and an rc.sysvinit init script to maintain compatibility.

All of the Slackware /etc/rc.d/rc.* files are Bash shell scripts, and can be edited by hand. For networking you might want to try the netconfig utility first, though. It probably handles just about everything you'd want to do and it's pretty easy to use, though you'll need to edit /etc/resolv.conf by hand to add more than one nameserver.

If you're relatively new to Linux and you make modifications to your system, you probably want to make sure you have a boot floppy and make copies of any rc.* file you edit. When in doubt, I usually save files as rc.*.old and make them unexecutable.

**Working with a running system**
Okay, so you've got a running system but you need to work on something in single user mode -- how do you get there? The next part of the article will explain how to change runlevels while a system is running instead of rebooting to change the runlevel, and why and when you should do it.

**The telinit command: Changing runlevels on a running system**
The telinit command is the way to change runlevels. When you execute telinit S as root (or whatever runlevel you'd like to change to), it changes runlevels, shuts down the prior runlevel, and then starts up the next one.

In a way, you are rebooting parts of the system. However, the ability to shutdown and restart services is one of Linux's most endearing qualities. Need to change the IP address for your machine? No problem, make a few changes and restart your networking services. As long as everything is configured correctly you're back up and running so quickly it's hard to tell anything has changed. Try doing that on other operating systems where you have to reboot just because you've changed wallpaper on your desktop :)

The only time it's really necessary to reboot or completely shut down a Linux box is if you're adding or changing hardware, assuming you're working with a device that isn't hot-swappable, or if you've had a break in and need to take the machine offline to repair the damage. Unlike other operating systems, I've never seen rebooting solve a problem on a production system that couldn't be solved without rebooting. I have managed to hang non-production machines by playing around with commands like hdparm, but I expected it to happen.

Let's say you need to perform some system maintenance that requires having the system in single-user mode. For instance, tuning your hard drive with hdparm. The first step is to su to root.

Then we'll execute the telinit command to bring the system into single-user mode:

```
telinit S -t 60
```

The "-t" argument is optional; it tells telinit to wait 60 seconds before actually performing the switch to single-user mode. However, as soon as the command is executed, anyone who is logged into the machine will get a warning on their console that the system is going to switch runlevels or go down in 60 seconds.

When the 60 seconds are up, init shuts down the processes that aren't used in the single-user mode and brings the system back up in single-user only mode. You'll then be prompted for the root password to perform system maintenance.

The process the system uses to get into single user mode is a bit different. The default for single user mode calls for init to invoke the sulogin command on the console and requiring a root login to work in single user mode.

After the system enters single user mode you should see a message like this:

```
        Give root password for system maintenance
        (or type Control-D for normal startup):
```

Once you've performed your maintenance you can bring the system back up by executing this command:

```
telinit 3
```

This tells the system to re-enter a multi-user runlevel. You could substitute "2" or "4" for "3" in this command. On Slackware systems runlevel 4 will put you in multi-user mode with one of the X display managers, so you will log directly into X.

If you have a UPS hooked up to your system that has a serial cable, it's possible to have the UPS send your system a signal in the event that power goes out. This is very useful if you have a production-class system with a large filesystem. I've seen what happens when a 100GB RAID ext2 filesystem is not cleanly unmounted (fsck takes about four hours to complete). On the other hand, a properly configured UPS can alert the system of the power outage and send telinit/init the SIGPWR signal, which will cause init to bring the system into single-user mode or shut the system down altogether, depending on how it's configured.

**Shutdown**
Okay, so you're sick of playing on the computer and you're going to venture out into the Big Blue Room for a while. Hopefully you already know it's a no-no to just hit the power switch when you're done, but you might not know all of the ways you can shut your system down.

In Linux you can reboot the system with the Three-Finger Salute: the Ctrl+Alt+Del key sequence sends the system the message to go through the shutdown process and restart. That is, unless you tell it not to.

If you'd like to disable the key sequence, there's a line in the inittab that you need to comment out:

**Partial Slackware inittab**

```
# Script to run when going multi user.
rc:2345:wait:/etc/rc.d/rc.M

# What to do at the "Three Finger Salute".
#ca::ctrlaltdel:/sbin/shutdown -t5 -rf now

Comment out the above line to disable Ctrl+Alt+Del hotkey.
```

What happens when you shut down? The shutdown command is called to bring the system down in an orderly fashion. However, shutdown doesn't do all the work itself, it signals init that it's time to go into runlevel 0, 1, or 6.

The shutdown command also notifies all users that are logged into the system that the machine is going to shut down. The login command is blocked after that so no one else can start a session.

To use the shutdown command to bring a Linux system down, use this command:

**Shutting down**

```
shutdown -h now
```

If you want to give users time to log off and save files, use this command:

```
shutdown -h -t 60
```

The "-h" switch for the shutdown command tells the system to shut down completely after halt. If you have APM enabled in the kernel, it may power your machine off for you, otherwise it's safe to hit the off switch at this point.

The "-t" switch is the time in seconds it takes until the system begins to shut down. If you want to bail out on a shutdown, that's possible too. To stop a pending shutdown, type:

```
shutdown -c
```

That will cancel any previous shutdown command. If, for some reason, you don't want to shut the system down but you want to send a warning to users anyway that the system is shutting down, use the "-k" argument with the shutdown command.

Knowing how to use telinit and init can come in handy when you're making modifications to your Linux system. Here we've covered the basics of changing runlevels and finding your way around the init scripts for the Slackware Linux distribution. The directory structures and locations of the files are different for each distribution, but after reading this article you should be able to figure out your system's init scripts even if you're not using Slackware Linux.

**Resources**

- Visit the Slackware Linux home page.
- Order Install, Configure, and Customize Slackware Linux, a beginner's guide to Slackware, written by Joe Brockmeier and others.
- For additional background, read "How Your Computer Boots".
- Read more about the design differences in Berkeley Software Distribution (BSD) UNIX and AT&T System V UNIX, and Vi vs. Emacs, one of the holy wars of computing.

**About the author**

Joe "Zonker" Brockmeier has been working with Linux since 1996, and writing about it nearly as long. He is a Senior Editor with User Friendly Media and a contributing editor for Linux Magazine, Enterprise Linux Magazine, and Unix Review. Joe is a refugee from radio broadcasting with degrees in English and communications/journalism who likes to spend spare time caring for his menagerie of 11 (!) computers and watching gangster movies. You can contact him at jbrockmeier@earthlink.net.

**What do you think of this article?**

Killer! (5) Good stuff (4) So-so; not bad (3) Needs work (2) Lame! (1)

**Comments?**

Submit feedback