Tuning Red Hat for maximum performance

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

Section 1. About this tutorial

What does this tutorial cover?

This tutorial details the process involved in transforming a stock, "out of the box" Red Hat installation into a finely tuned, stable system customized to individual needs and tastes. The material presented here is based on Red Hat 7.3, although many of the techniques and procedures discussed are equally applicable to other mainstream Linux distributions. And while the title uses the phrase "performance tuning," you'll soon discover that performance and security often go hand in hand. Topics addressed include:

Who should take this tutorial?

This tutorial is written for Linux server administrators who want to customize Linux to increase the performance and reliability of an installation.

Readers should have a basic understanding of common *NIX administrative concepts and tasks, such as file system layout, daemons and services, editing configuration files, the Linux initialization process, and networking.

Prerequisites

To achieve maximum benefit from the material presented in this tutorial, readers should have a basic understanding of common *NIX administrative concepts and tasks, such as file system layout, daemons and services, editing configuration files, the Linux initialization process, and networking.

Readers wishing to follow along with the examples presented should have at their disposal a pre-existing Linux installation, or a system on which there is enough available free drive space to install a rudimentary Linux distribution. As noted, while this tutorial is based on Red Hat 7.3, the concepts discussed are applicable to a wide range of other Linux products.

About the author

Over the last several years, Tom Syroid has brought the skills and hands-on experience gained from a variety of professions to the task of technical writing; he is the author of *Outlook 2000 in a Nutshell* (O'Reilly) and *OpenLinux Secrets* (Hungry Minds). He has also written numerous articles on Linux/UNIX system administration and security issues. Tom is proficient in the nuances of Samba, Apache, Microsoft Office (97, 2000, and XP), SSH, DNS/BIND, and various other Open Source products/technologies. On the operating front, Tom has configured and administered systems running all variants of Windows (95, 98, ME, NT, 2000, XP), most Linux distributions (Red Hat, OpenLinux, Mandrake, Slackware, TurboLinux, SuSE, and Yellow Dog Linux), and AIX (4.3.x, and 5L). Tom's hardware capabilities are equally eclectic: he has experience in spec'ing, building, and maintaining all shapes and sizes of PCs, as well as

considerable background in IBM's RS/6000 product line, particularly the F50 series.

Tom welcomes comments and feedback. You can reach him at *tom@syroidmanor.com*.

Section 2. First things first

Why tune a stock installation?

Red Hat Linux is one of the most popular Linux distributions on the market. But the fact that Red Hat is designed to install and run on a wide range of systems and architectures implies compromise. The demands placed on a multi-processor, high-load Web server are very different from those placed on a single processor, mid-range notebook. By learning to customize Linux to their unique needs, users can -- with relative ease -- dramatically increase the performance and reliability of an installation.

Installation

While it's likely you already have an existing Linux installation, in the interests of completeness, let's begin with the basics and step through a "clean," "from scratch" installation of Red Hat 7.3.

The current crop of Linux installers make child's play out of a process that used to be hit-and-miss and fraught with potential pitfalls. Now you simply click through a few setup screens, provide some rudimentary information, and supply the necessary CDs when requested. There are three key choices to make during installation, however, that can potentially affect both performance and ease-of-maintenance down the road: Partition layout, the underlying filesystem, and packages installed.

First up, partitioning considerations.

Partitioning considerations

When deciding how to divide up a hard drive, you have three key things to consider:

- How much drive space do you have to work with?
- How granular do you want your filesystems?
- How often do you anticipate upgrading or reinstalling the operating system?

Available space will obviously determine how big to make each partition. In considering the issue of granularity, do you prefer one monolithic filesystem, or smaller partitions? If you choose to let Red Hat automatically partition your drive, it will create a small `/boot` (25MB) partition and allocate the balance of the hard drive's space to `/` .

The disadvantage of this approach is that if you decide six months down the road to change this layout, you're going to have to start over and install from scratch again. If the system has been partitioned with several smaller partitions, however, data can be moved to another partition, the original can be destroyed and resized, and the data moved back. Last, but not least, if your system is destined to be a development box and subject to frequent clean

installations in the course of testing programs or operating system configurations, go for multiple partitions. Then when you reinstall, it's simply a matter of formatting the filesystems where operating system-specific files reside, and keeping intact any filesystems containing user data, source code, etc.

The following is a suggested multiple-partition layout that's not overly complex, but allows for maximum flexibility (filesystem sizes are based on a 20 GB drive and 512 MB of RAM; adjust to suit your needs and drive size):

```
/boot (25MB): hda1 swap (1000MB): hda2 / (3000MB): hda3 extended (remaining drivespace):
hda4 /var (3000MB): hda5 /usr/local (3000): hda6 /home (balance remaining): hda7
```

Choosing a filesystem

In conjunction with the process of partitioning a drive, you'll also have to decide what filesystem to put on each partition.

Current trends favor some form of journaling filesystem primarily due to today's hard drive sizes. It can take in excess of a half-hour to fully check (and, if necessary, repair) a large hard drive, and if your circumstances demand a server with 24x7 availability, a journaling filesystem can cut this time to less than a minute. But depending on whom you talk to, journaling filesystems are "not ready for prime time" and can potentially corrupt data beyond repair (on a personal note, I've used EXT3 filesystems on all my machines for over a year now, and have never had a moment's trouble. As the saying goes, however, Your Mileage May Vary [YMMV]).

Red Hat 7.3 offers these Linux filesystem types: EXT2 (non-journaling), EXT3 (journaling), VFAT (readable directly by a Windows installation), software RAID, and swap.

As noted, the choice of a filesystem(s) will depend on personal experience, the system (dual boot, or single OS), and the machine's purpose. Note that it is perfectly acceptable to mix and match filesystems. For example, ReiserFS can only be installed on a partition at least 50 MB in size due to the amount of metadata that accompanies the filesystem. So using the partitioning layout example from the preceding panel in conjunction with Reiser means that the `/boot` partition would have to be larger or formatted EXT2.

Gathering and configuring the necessary pieces to install an alternate journaling filesystem (for example, ReiserFS or JFS) not packaged with Red Hat is beyond the scope of this tutorial. Check the *Red Hat Web site* for the filesystem in question; they typically provide step-by-step installation instructions specific to a given distribution. See the Resources on page 19 section for links.

Package selection

The key with package selection is to install only the packages you intend to use. Installing packages that will never get used has several costs:

In broad terms, there are three routes to package installation under Red Hat 7.3 (earlier versions vary slightly in the choices offered). The first package selection screen allows you to choose to upgrade a system or a clean installation. If the upgrade path is chosen, Red Hat will scour the hard drive for a previous installation and upgrade packages as appropriate.

If the latter is chosen, you can then choose a package "group"; in other words, workstation, laptop, server, or custom. Each selection installs a pre-configured group of packages geared to the system's intended use. Workstation, laptop, and server are self-explanatory; custom provides more detailed authentication options (NIS, LDAP, or SMB) as well as a longer list of package choices later in the installation process.

Once again, individual needs will dictate your choices, but here are two suggestions from someone who's done literally hundreds of Linux installations and messed up his share: (1) Whenever possible, install clean; trying to upgrade a system adds a layer of complexity that can potentially break more than it fixes, and (2) unless you have a specific need for a customized installation, make broad selections using the package groupings provided (workstation, laptop, or server) and cull/tweak package selections later. Removing unused packages is fully detailed later in Culling unused packages on page 7 .

Installation addendum

Red Hat has a relatively foolproof installation procedure that works as advertised at least 95% of the time. If you happen to fall into the unfortunate 5%, here are some pointers:

Assuming your installation is completed successfully, and you can log into the system, spend some time exploring and generally testing to ensure everything works as advertised. If something is grossly amiss at this stage of the game, it might be easier to do a clean install to resolve the problem. The goal here is to begin with a solid, stable system and tweak from a "known good" starting point.

In the next section, Securing and tweaking installed packages on page 7 , we begin this tweaking process by updating all key packages to account for any bug fixes or enhancements released by your distribution.

Section 3. Securing and tweaking installed packages

Bolstering the integrity of your installation

This section details the next logical step in updating and optimizing your Linux installation: updating your installed packages to account for any errata or bug fixes released since the distribution was shipped and removing any installed packages that will not be used. And although the details discussed here are Red Hat-specific in nature, all distributions have similar mechanisms for package management and update. See your vendor's site for details.

The process of updating and culling a Linux installation involves two distinct tasks:

Which step you complete first is of no real consequence as you are obviously not going to update a package you don't intend to keep on the system.

The reason for applying errata and bugfixes is pretty obvious. Your system's integrity and reliability are directly proportional to the programs installed there. The reason for removing all unneeded packages has already been touched upon -- there's no point in clogging up an installation with programs that will never be used and draining systems resources by running services that no one needs.

The Red Hat Package Manager

The Red Hat system for installing, maintaining, and managing programs is call the Red Hat Package Manager, or RPM. It's simple to use once you've mastered a handful of basic commands (there are also a handful of graphic frontends available for Linux; for the purposes of this tutorial, however, we'll stick to the command line because it's generic to all distributions).

The first step is to determine what packages are actually installed on your system. To query [q] the RPM database for all [a] packages installed, the command is:

```
[tom@thor  tom]  # rpm -qa
```

The result list is going to scroll by pretty fast, so a better solution is to redirect the query to a text file that you can open in a text editor or print out for reference.

```
[tom@thor  tom] # rpm -qa > package-list
```

Culling unused packages on page 7 details the removal of non-essential packages.

Culling unused packages

With the package list generated in The Red Hat Package Manager on page 7 in hand, the

balance of the process is straightforward. Select which packages to delete, and one at a time, remove them from the system using the following command. Note that you'll need to be logged in as user *root* to remove or install packages:

```
[root@thor  root] #  rpm -e package-name
```

If removing a package will cause another package to "break" (that is, one package has a "dependency" on another's code), the system warns of the problem and will not complete the command. If this happens, you'll have to decide whether to delete the dependency or not (which, in turn, might have yet another dependency -- welcome to the world of Linux!).

To get a detailed description of an installed package (including name, version, who built the package, etc.), type **`rpm -qi package-name`**.

Adding new packages

The flip side of removing any non-essential packages is adding any required packages that were not installed by Red Hat's default package selections. One good example is the Lynx text browser, a staple for all admins who prefer the command line over a GUI desktop.

The procedure for installing packages (either from Red Hat CDs or another source such as rpmfind.net (see Resources on page 19) is equally straightforward. Here is the procedure to follow when using the Red Hat CDs as a source:

```
[root@thor  root] #  mount /dev/cdrom
[root@thor  root] #  cd /mnt/cdrom/RedHat/RPMS
[root@thor  root] #  rpm -ivh package-name
```

RPMs from outside sources will have to first be downloaded, of course, and then installed using the above `rpm -ivh ...` command. For a complete list of rpm options, type **`man rpm`** at the command prompt.

Updating installed packages

With your installed package list cleaned up, the next step is to get online, go to the Red Hat *errata* page, and scan through the list of available updates. Technically, there are three distinct update pages: A security alerts page, a bugfix page, and an enhancements page. The distinction between them lies in the level or importance of the update. Security alerts are updates that relate directly to a program's ability to function in a safe and secure manner; the security updates page is a "must" to monitor on a routine basis, especially for a system connected directly to the Internet. The bugfix page is -- as the name implies -- package updates that correct program "bugs." Finally, the enhancements page contains a list of packages that have been updated following program enhancements. Ensure you select the correct errata page for your distribution version, as some updates apply only to a specific release.

Remember the *rpm -qa* list discussed in The Red Hat Package Manager on page 7 ? Now would be a good time to generate a new one, so you can compare existing installed packages to what's new on the errata lists.

Next, we'll look at installing the updated packages.

Installing the updated packages

When you have all the required packages downloaded as described in Updating installed packages on page 8 , type:

```
[root@thor  root] #  rpm -Uvh package-name
```

to install them. Note that you can install "groups" of related packages by supplying the package name without any version information. For example, many programs have a core package, a client package, and a server package (LDAP, Samba, etc.). To install all required Samba packages, the command would be **`rpm -Uvh samba`**. Note that clicking on a package on the errata list brings up the package page containing an explanation for the update, any relevant dependencies, and complete installation instructions.

Next up, it's time to tend to some miscellaneous housekeeping chores: turning off all unnecessary services, cleaning up the user and group files, shutting off some virtual consoles, and some brief comments on X-Windows.

Section 4. Miscellaneous housekeeping

System services administration

With package maintenance and upgrades out of the way, it's time to turn to the system itself and perform some further cleanup. Taken individually, the steps detailed in this section do not amount to much. But taken together, they do add up to a well-tuned, easily maintained system.

System services are programs, typically in the form of a *daemon*, that listen in the background for requests on a preassigned TCP/IP port. When a user connects to a Web server, that service request is answered by the Apache daemon listening on (by default) port 80. If the request is valid, Apache responds by sending the requested Web page back to the client and then becomes "dormant" again until another request comes along. Granted, a dormant service does not require much in the way of system resources, but why have Apache running at all if your system is not configured to be a Web server? Why have the `portmapper` daemon active if you never use NFS? Based on the "less is better" philosophy of this tutorial, there is no logical reason to have any services running that are not specifically needed by a given system.

Deselecting system services

On a Red Hat system, to control which services run at startup, login as root (or get a full root environment by typing **`su -`**) and enter:

```
[root@thor  root] #  setup
```

This will invoke a curses-based administrative setup program. Next select *System services* for the list provided. Scroll down the entries there and uncheck all unnecessary services. To see the results, reboot the system and examine the output of the command **`netstat -lnp --ip`**. Below are "before" (stock system install) and "after" (with all unused services turned off, leaving just the SSH daemon running):

```
[root@thor root]# netstat -lnp --ip
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address   Foreign Address  State   PID/Program name
tcp      0      0   0.0.0.0:32768   0.0.0.0:*        LISTEN  669/rpc.statd
tcp      0      0   0.0.0.0:111     0.0.0.0:*        LISTEN  641/portmap
tcp      0      0   0.0.0.0:22      0.0.0.0:*        LISTEN  831/sshd
tcp      0      0   127.0.0.1:25    0.0.0.0:*        LISTEN  906/sendmail: accep
udp      0      0   0.0.0.0:32768   0.0.0.0:*                669/rpc.statd
udp      0      0   0.0.0.0:111     0.0.0.0:*                641/portmap

[after]
[root@thor root]# netstat -lnp --ip
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address   Foreign Address  State   PID/Program name
tcp      0      0   0.0.0.0:22      0.0.0.0:*        LISTEN  712/sshd
[root@thor root]#
```

Note: The '-l' option is preferred over the '-a' (all) option as we're interested only in daemons *listening* on a socket.

Trimming back virtual consoles

Virtual consoles provide a way to log in to a system multiple times as either the same user or another user. To start a new console or login session, hold down the <CTRL> and <ALT> keys, and press a function key F1 - F6. Administrators typically use multiple virtual consoles to perform a task as root, while retaining their original login session as a regular user. This is all well and good, but how many people need to access six virtual consoles at once (remember, virtual consoles are similar to system services; they remain active, using system services, until activated). To reduce the number of virtual consoles available, open `/etc/inittab` and comment out one or more lines referencing `/sbin/mingetty tty`*x*. Below is the output of just such an edit:

```
# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
#4:2345:respawn:/sbin/mingetty tty4
#5:2345:respawn:/sbin/mingetty tty5
#6:2345:respawn:/sbin/mingetty tty6
```

Disabling X auto start

Most administrators tend to do a lot of routine tasks from the command line. Most commercial Linux distributions, however, default to starting an X-Window manager following system initialization. That's a lot of system resources to have running if you're compiling a program or editing configuration files. To change this behavior, edit the `/etc/inittab` file and locate the line that reads: `id:5:initdefault` and change it to `id:3:initdefault`. The system will start with a command line login, and when the need arises to run a window manager, it's easy to simply type **`startx`**.

While we're on the topic of window managers and, by association, desktop environments, enormous performance improvements can be gained by recompiling your desktop environment with the latest version of `binutils` (as of this writing, 2.12.x) and the gcc 3.x compiler. The 2.12.x version of `binutils` enables the compiler to build programs with a feature called *combreloc* permitting pre-linking against relocated code: something common in most C++ code, and prominent in Qt and KDE. Such a large undertaking is definitely not for the faint of heart, however. Depending on the machine, Qt and KDE alone require four or five hours to compile. The end result -- for those with patience and the requisite stamina -- are larger executables that run significantly faster.

For details (and caveats) on the process, search one of the KDE newsgroups (see Resources on page 19).

Hard drive tuning

If your system is equipped with an IDE hard driver, and you're running an older kernel (the newer 2.4.x kernels typically optimize hard drive performance settings without additional intervention), the program `/sbin/hdparm` bears investigating. Be warned however, improper use of `hdparm` can at a minimum lead to a *denigration* in hard drive performance, and in extreme cases, damage the drive itself. Caveat Emptor, and read the *man* page **`man hdparm`** carefully.

Some of the more common options used with `hdparm` are:

`/sbin/hdparm -c1 /dev/hda` ***`or hdb or hdc, etc.`***: Turns on 32bit I/O on the PCI bus.

`/sbin/hdparm -d1 /dev/hda`: Enable DMA.

`/sbin/hdparm -d1 -X66 /dev/hda`: Enable UltraDMA transfers.

To get a listing of current hard drive setting on your system, (as root) type: **`/sbin/hdparm /dev/hda`**

Once you have everything tuned to perfection, add the various commands to the `/etc/rc.d/rc.local` file so they're run every time the system boots.

Tweaking TCP/IP resources

Tweaking the TCP/IP stack is a black art for most people, but if you run a heavily loaded Web server for example, selectively changing some of Red Hat's default values can lead to a significantly more responsive system. TCP/IP configuration values are store in the `/proc/sys/net/ipv4` directory and typically accept a value or are turned on or off with "0" (off) or "1" (on). At the time of this writing, Red Hat 7.3 was shipping with the following defaults:

- tcp_fin_timeout 180
- tcp_keepalive_time 7200
- tcp_window_scaling 1
- tcp_sack 1
- tcp_timestamps 1

The changing the values below will increase the number of TCP/IP connections the server can handle, lower the amount of time the server waits before killing a stale connection, and turn off several non-essential IP extensions. Add these settings to your `/etc/sysctl.conf` file, and restart the network daemon (**`/etc/rc.d/init.d/network restart`**).

```
# Decrease the time default value for tcp_fin_timeout connection
net.ipv4.tcp_fin_timeout = 30
# Decrease the time default value for tcp_keepalive_time connection
net.ipv4.tcp_keepalive_time = 1800
# Turn off tcp_window_scaling
net.ipv4.tcp_window_scaling = 0
# Turn off the tcp_sack
net.ipv4.tcp_sack = 0
#Turn off tcp_timestamps
net.ipv4.tcp_timestamps = 0
```

Section 5. Compiling a custom kernel

Why a custom kernel?

The heart and soul of any Linux system is the kernel itself. This section addresses the topic of compiling a custom kernel, why you might want to take the custom approach, the two basic approaches to kernel building, the steps involved in compiling a kernel, and various options to be aware of if and when you do so.

Generally speaking, long-time Linux users rarely run stock kernels on their systems. It's not that there's anything fundamentally wrong with the kernels provided with most distributions -- as a matter of fact, Red Hat is well-known for shipping stable, well-engineered kernels with their products. The problem is, all major distributions have to compile their kernels to run reliably on the widest range of architectures and systems as possible. Given this fact, what's "good enough" for the masses is almost certainly not optimized for your needs and purposes.

Some of the more common reasons for compiling a custom kernel are:

Probably the biggest reason more people don't compile their own kernel is a general perception that the process is intimidating or difficult. Hopefully, by following the guidelines detailed in this section, you'll find the opposite to be true.

Two schools of thought...

There are two basic schools of thought when it comes to compiling a kernel: *monolithic* and *module-based* .

As with most things Linux, common practice draws a little from both camps. Key services and drivers are compiled into the kernel, and less-used components are built as modules. As a rule of thumb, kernels built for use on one specific system lean more toward the monolithic philosophy. Kernels built to run on more than one system favor the module-based approach.

Kernel compilation overview, Part 1

Building a kernel is not as difficult as it's often made out to be as long as you're careful to follow certain steps. In a nutshell, here are the steps involved:

- Decide on a kernel build.
- Download the required source tarball and move it to `/usr/src`.
- 'cd' to `/usr/src`.
- For safety's sake, remove the existing symbolic link to the current kernel source (**`rm -rf /usr/src/linux`**). Current kernel packages unpack to a directory named `/usr/src/linux-`*`kernel-version`*, but some older packages still use the old convention of uncompressing to `/usr/src/linux` -- which effectively overwrites your

existing Linux source tree.

- Uncompress the source code package (**`tar xvzf linux-`*`kernel-version`***).
- Recreate the symbolic link previously removed (**`ln -sf /usr/src/linux-`*`kernel-version`*` linux`**).
- 'cd' to `/usr/src/linux`.
- Type **`make mrproper`**. This cleans up the source tree and removes any old object files. It will also remove the existing `.config` file (where your compile options are saved), so if you want to save these settings, copy this file to a safe location and then copy it back to `/usr/src/linux` when the make process completes.

The next panel completes the process.

Kernel compilation overview, Part 2

In the previous panel, we began the process of building a kernel by downloading and uncompressing the source code and cleaning up the source tree. Here we complete the process of building the kernel.

- Type `make config, make menuconfig, make xconfig`, or `make oldconfig` depending on your interface preferences. `make config` is a console-based configuration process, `make menuconfig` provides a cursed-based menu configuration, `make xconfig` is an X-windows interface, and `make oldconfig` can be used when you want to use an existing kernel configuration to built a new kernel and only want to see new options present in the new kernel tree.
- When you're satisfied with your configuration choices, type **`make dep ; make clean`**.
- To build the actual kernel, type **`make bzImage`** or `make install`. The first command builds the kernel image but doesn't physically install it in the `/boot` directory; the latter command builds and installs the kernel image for you.
- If the kernel compilation completes with error, type `make modules && make modules_install`. This builds and installs any module-based components.
- Finally, copy the new kernel to `/boot` (if you used the `make bzImage` command). If you're running a Red Hat-based distribution, you'll also have to update the `System.map` and `initrd` image. Be sure to add a new entry to your boot loader configuration so if the kernel fails to boot for some reason, you can still get back into your system and make the appropriate corrections.

A comprehensive HOWTO detailing all the ins-and-outs of kernel compilation is available from Red Hat (see Resources on page 19). If you're new to building Linux kernels, it's worth a read.

Configuration tips and tricks

The most important step in building a finely tuned Linux kernel is the aforementioned `make configure/menuconfig/xconf/oldconfig` process. This is where you choose what

systems, services, and drivers to include in the kernel, and which to build as modules. The most common questions raised regarding this process are "what do I include and what do I exclude from the configuration," and "which components should I build into the kernel and which components should I build as modules."

The short answer to the first question is "only what you need to make your system function as desired" (remember, less is more). A second, and perhaps more convoluted answer would be, "trial and error." Even experienced administrators don't always get the configuration right the first time. When the compilation process fails or you end up with a non-bootable kernel, go back into the system using your backup boot loader entry, and double-check the configuration; try again with a different set of options.

The answer to the second question regarding what components to build into the kernel and what to build as modules very much depends on how you'll be using the system. That and experience. For example, the Maestro3 sound driver does not work on some Dell notebooks when compiled into the kernel -- for some reason, this particular driver wants to load as a module. On other systems with more mainstream hardware components, you can build just about anything you want into the kernel. Again, system configuration and experience often dictate your choices here.

Key options in kernel configuration

In an attempt to remove at least some of the magic from the kernel configuration process, the next several panels detail some of the key options to take note of, and where applicable, how they detract from or enhance system performance.

Before we begin with specifics, note that when using the `menuconfig` and `xconfig` configuration interfaces, descriptions are available for most options by selecting the item in question, tabbing to the help button, and hitting Enter.

Also, keep in mind that configuration options vary from kernel version to kernel version. If you don't see a particular option and want that functionality in your kernel, you have two choices: Try a different kernel, or find the appropriate *diff* file and manually patch the kernel. For an explanation of how to apply a patch, see the previously mentioned Kernel-HOWTO file (see Resources on page 19).

Kernel configuration options: Code maturity and level options

The very first option to be aware of is the first menu item on the list: *Code maturity level options*. Unless you select "Prompt for development and/or incomplete code/drivers", many kernel configuration options will remain hidden from view.

The defaults under *General setup* and *Loadable module support* suffice for most systems.

Under *Processor type and features*, pay attention to the type of processor the kernel is compiled for (the default is Pentium 3/Celeron). SMP is selected by default; deselect this option if you're building on a single processor system. Current kernels offer support for a "Preempible Kernel." Select this option. According to the feature description, this "reduces the latency of the kernel when reacting to real-time or interactive events by allowing a low priority process to be preempted even if it is in kernel mode executing a system call. This allows applications to run more reliably even when the system is under load."

Kernel configuration options: Power management options

Moving down the list, the next menu group to pay attention to is *Power management options*. Laptop users generally desire at least some form of power management, but the interaction between BIOS and operating system in this regard is a well-known source of problems. Unless you're confident your system fully supports ACPI, stick with simple APM. Enabling ACPI on all but a few select notebooks is a sure way to build a kernel that locks hard on bootup. For desktop users and server administrators, your need for power management will end up being a "personal preference" call.

The *Bus options* entry contains support for various bus topologies including PCI, EISA, MCA, and PCI hotplug support. Again, defaults are in order here with one notable exception. If you're using a PCMCIA network or wireless card and plan to use the `pcmcia-cs` driver code as opposed to the drivers provided by the Linux kernel, *disable* the PCMCIA/Cardbus support option here but leave *PCMCIA network device support* enabled under the *Network device support* menu.

Kernel configuration options: Device support options

For the most part, all device support options on the configuration menu are self-explanatory: If you want support for a given option, select it; if you don't, leave it out. One notable exception is *Network device support*. The current 2.5.x development kernels enable a small handful of devices by default, but some older kernels enable numerous devices by default (as modules). Scour the options here with care and deselect all entries you know will never be present. It's a good idea to build into the kernel one or both EtherExpressPro/100 options as a large number of generic network cards support this chipset, and a fallback network device option is always a good thing to have around.

The *Filesystems* menu is also another place to make selections with care. If you opted for ReiserFS or EXT3 filesystems, the appropriate support *must* be enabled and built into the kernel; failure to do so will leave you with a system that won't boot (Linux has to be able to access the filesystems on all partitions at system initialization). It's also a good idea to leave the EXT2 option selected. The code involved is relatively small and is also good to have in the kernel should problems arise. Under *Filesystems-->Network files systems*, be sure to select the SMB option if you plan on connecting to Windows share via Samba (see Resources on page 19 for a link to the samba.org homepage).

Kernel configuration options: Sound driver options

Finally, play close attention to the options you select under *Sound*. Sound drivers under Linux are notoriously buggy and temperamental. Choosing the correct driver can mean the difference between fighting for hours with configuration issues and having a device that works with minimal fuss.

Building a custom Linux kernel is certainly not rocket science, but it does take a certain amount of patience and trial and error. In the end, however, tenacity will be rewarded.

Section 6. Summary, resources, and feedback

Summary

Optimizing a stock Red Hat installation is not difficult or even that time consuming; it's simply a matter of understanding the underlying components that comprise a given distribution and leveraging Linux's inherent configurability to fine-tune those components.

This tutorial focused on four key areas:

- Installation issues
- Removing, installing, and updating program packages
- Various system tweaks to increase performance hardware and software performance
- The steps required to build a custom kernel

Resources

Given Red Hat's popularity and pervasiveness, there's no shortage of information available on the Web relating to performance tuning, partitioning options, filesystem options, general system administration, or building a custom kernel. Below are several additional references for your browsing pleasure. And don't forget the power of *Google* when searching for answers to seemingly obscure questions.

Red Hat-specific information:

- The definitive stop for all things Red Hat is, of course, the *Red Hat Web site*. There you'll find *product resources and support docs*, *company-supported mailing lists* , the *Red Hat errata page*, and a *current hardware compatibility list*.
- *Securing and Optimizing Linux (Red Hat Edition - A Hands on Guide)* is an open source online book project written by Gerhard Mourani. Some of the material is somewhat dated, but most of the concepts discussed do not change across version releases.

The Linux Documentation Project (LDP)

One of the best "one-stop-shops" for generic Linux documentation is the *Linux Documentation Project*, which is, incidently, sponsored in large part by Red Hat. The LDP contains thousands of HOWTOs, guides, FAQs, and even a searchable listing of man pages. If you can't find it here, it's not written yet.

Resources, continued

Kernel source and documentation

- *Linux kernel source* are at kernel.org. At the top of the index page is a listing of all current kernels (development and stable), plus links to all important *changelog* documents that detail the updates and bug fixes in each kernel release.
- Kerneltrap.org is a good source for *daily news relating to the Linux kernel*.
- *Kernel Cousins* is described as an "attempt to track the activities of various developer mailing lists. By summarizing the discussions that take place, a much finer grain of information can be provided than a normal news service."
- For all the ins-and-outs of kernel compilation, consult this *comprehensive kernel HOWTO*.

Filesystem information

Further information on current filesystems can be found at the various supporting Web sites:

- Find white papers, FAQ, man pages, utilities, source code, and installation documentation for *JFS for Linux* at *developerWorks*.
- Find similar resources for the *ReiserFS filesystem* at namesys.com.
- Information on the EXT3 filesystem is scattered everywhere. Red Hat has a *white Paper on EXT3*, and Daniel Robbins has written two excellent *developerWorks* articles on EXT3; the first *introduces ext3*, and the second describes *surprises in ext3*.
- For a good overview of filesystem issues and choices in general, start at Part 1 of Daniel's *Advanced filesystem implementor's guide* and work your way through all 11 parts.

Miscellaneous resources

- Read about an *alternate method for adding Red Hat packages*.
- For information on Qt and KDE, visit the *Qt and KDE mailing lists* at kde.org.
- Find *comprehensive Samba-related resources* at samba.org. Also check out Daniel Robbins' Samba series on *developerWorks*: *Part 1 on key concepts*, *Part 2 on setup*, and *Part 3 on configuration*.

Your feedback

We look forward to getting your feedback on this tutorial. You may also contact the author directly at *tom@syroidmanor.com*.

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our

production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial Building tutorials with the Toot-O-Matic demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11. We'd love to know what you think about the tool.