

LPI certification 102 exam prep, Part 3

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.

1. About this tutorial	2
2. TCP/IP networking	4
3. Internet services	8
4. Security overview	13
5. Printing	22
6. Resources and feedback	28

Section 1. About this tutorial

What does this tutorial cover?

Welcome to "Networking," the third of four tutorials designed to prepare you for the Linux Professional Institute's 102 exam. In this tutorial, we'll introduce you to TCP/IP and Ethernet Linux networking fundamentals, show you how to use the `inetd` and `xinetd` superservers, provide you with important tips for securing your Linux systems, and also show you how to set up and use a Linux print server. By the end of this series of tutorials (eight in all; this is part seven), you'll have the knowledge you need to become a Linux Systems Administrator and will be ready to attain an LPIC Level 1 certification from the Linux Professional Institute if you so choose.

The LPI logo is a trademark of the [Linux Professional Institute](#).

Should I take this tutorial?

This tutorial is ideal for those who want to learn about or improve their basic Linux networking and security skills. It's especially appropriate for those who will be setting up applications on Linux servers or desktops. For many, much of this material will be new, but more experienced Linux users may find this tutorial to be a great way of rounding out their important Linux system administration skills.

If you are new to Linux, we recommend that you first complete the previous tutorials in the LPI certification 101 and 102 exam prep series before continuing:

- [101 series, Part 1: Linux fundamentals](#)
- [101 series, Part 2: Basic administration](#)
- [101 series, Part 3: Intermediate administration](#)
- [101 series, Part 4: Advanced administration](#)
- [102 series, Part 1: Compiling sources and managing packages](#)
- [102 series, Part 2: Compiling and configuring the kernel](#)

About the authors

For technical questions about the content of this tutorial, contact the authors:

- Daniel Robbins, at drobbins@gentoo.org
- Chris Houser, at chouser@gentoo.org
- Aron Griffis, at agriffis@gentoo.org

Residing in Albuquerque, New Mexico, Daniel Robbins is the President/CEO of Gentoo Technologies, Inc., the creator of [Gentoo Linux](#), an advanced Linux for the PC, and the **Portage** system, a next-generation ports system for Linux. He has also served as a contributing author for the Macmillan books *Caldera OpenLinux Unleashed*, *SuSE Linux Unleashed*, and *Samba Unleashed*. Daniel has been involved with computers in some fashion since the second grade, when he was first exposed to the Logo programming

language as well as to a potentially dangerous dose of Pac Man. This probably explains why he has since served as a Lead Graphic Artist at **SONY Electronic Publishing/Psygnosis**. Daniel enjoys spending time with his wife, Mary, and their daughter, Hadassah.

Chris Houser, known to many of his friends as "Chouser," has been a Unix proponent since 1994 when he joined the administration team for the computer science network at Taylor University in Indiana -- where he earned his Bachelor's degree in Computer Science and Mathematics. Since then, he has gone on to work in Web application programming, user interface design, professional video software support, and now Tru64 UNIX device driver programming at [Compaq](#). He has also contributed to various free software projects, most recently to [Gentoo Linux](#). He lives with his wife and two cats in New Hampshire.

Aron Griffis graduated from Taylor University with a degree in Computer Science and an award that proclaimed him to be the "Future Founder of a Utopian UNIX Commune." Working towards that goal, Aron is employed by [Compaq](#) writing network drivers for Tru64 UNIX, and spending his spare time plunking out tunes on the piano or developing [Gentoo Linux](#). He lives with his wife Amy (also a UNIX engineer) in Nashua, New Hampshire.

Section 2. TCP/IP networking

Introduction

Setting up an Ethernet-based Local Area Network (LAN) consisting of a bunch of Linux machines is a common and relatively simple task. Generally, all you need to do is make sure that your Linux systems have an Ethernet card of some kind installed in them. Then, connect the machines to a central Ethernet hub or switch using Ethernet cabling. If all your systems have support for their respective Ethernet card compiled into the kernel (as well as TCP/IP support), then they technically have everything they need to communicate over your new Ethernet LAN.

Ethernet alone isn't much fun

While you'll then have all the hardware and kernel support needed for your LAN to work, it won't do much. The vast majority of Linux applications and services don't exchange information using raw Ethernet packets, or *frames*. Instead, they use a higher-level protocol called TCP/IP. You've undoubtedly heard of TCP/IP -- it's the suite of protocols that forms the foundation of the Internet in general (hence the name, Transmission Control Protocol/Internet Protocol).

The solution: TCP/IP over Ethernet

The solution, then, is to configure your new Ethernet LAN so that it can exchange TCP/IP traffic. To understand how this works, we first need to understand a bit about Ethernet. On an Ethernet LAN, in particular, the Ethernet card in every machine has a unique hardware address. This hardware address is assigned to the card at the time of manufacture, and looks something like this:

```
00:01:02:CB:57:3C
```

Introducing IP addresses

These hardware addresses are used as unique addresses for individual systems on your Ethernet LAN. Using hardware addresses, one machine can, for example, send an Ethernet frame addressed to another machine. The problem with this approach is that TCP/IP-based communication uses a different kind of addressing scheme, using what are called IP addresses instead. IP addresses look something like this:

```
192.168.1.1
```

Associating an IP address with an Ethernet interface

In order to get your Ethernet LAN working for TCP/IP, you need some way of associating each machine's Ethernet card (and thus its hardware address) with an IP address. Fortunately, there's an easy way to associate an IP address with an Ethernet interface under

Linux. In fact, if you are currently using Ethernet with Linux, your distribution's system initialization scripts very likely have a command in them that looks something like this:

```
ifconfig eth0 192.168.1.1 broadcast 192.168.1.255 netmask 255.255.255.0
```

Above, the `ifconfig` command is used to associate `eth0` (and thus `eth0`'s hardware address) with the 192.168.1.1 IP address. In addition, various other IP-related information is specified, including a broadcast address (192.168.1.255) and a netmask (255.255.255.0). When this command completes, your `eth0` interface will be enabled and have an associated IP address.

Using ifconfig -a

You can view all network devices that are currently running by typing `ifconfig -a`, resulting in output that looks something like this:

```
eth0      Link encap:Ethernet  HWaddr 00:01:02:CB:57:3C
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
          Interrupt:5 Base address:0xc400

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1065 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1065 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:262542 (256.3 Kb)  TX bytes:262542 (256.3 Kb)
```

Above, you can see a configured `eth0` interface, as well as a configured `lo` (localhost) interface. The `lo` interface is a special virtual interface that's configured so that you can run TCP/IP applications locally, even without a network.

TCP/IP is working!

Once all of your network interfaces are brought up and associated with corresponding IP addresses, your Ethernet network can be used to carry TCP/IP traffic as well. The systems on your LAN can now address each other using IP addresses, and common commands such as `ping`, `telnet`, and `ssh` will work properly between your machines.

Name resolution limitations

However, while you'll be able to type things like `ping 192.168.1.1`, you won't be able to refer to your boxes by name. For example, you won't be able to type `ping mybox`. To do this, you need to set up a file called `/etc/hosts` on each of your Linux boxes. In this file, you specify an IP address, along with the name (or names) that are associated with each IP

address. So if I had a network with three nodes, my `/etc/hosts` file might look something like this:

```
127.0.0.1      localhost
192.168.1.1    mybox mybox.gentoo.org
192.168.1.2    testbox testbox.gentoo.org
192.168.1.3    mailbox mailbox.gentoo.org
```

Note that `/etc/hosts` contains an obligatory mapping of "localhost" to the 127.0.0.1 IP address. I've also specified the hostnames of all the systems on my LAN, including both their short name ("mybox") and their fully-qualified name ("mybox.gentoo.org"). After copying this `/etc/hosts` file to each of my systems, I'll now be able to refer to my systems by name, rather than simply by IP address: `ping mybox` will now work!

Using DNS

While this approach works for small LANs, it isn't very convenient for larger LANs with many systems on them. For such configurations, it's generally much better to store all your IP-to-hostname mapping information on a single machine, and set up what is called a "DNS server" (domain name service server) on it. Then, you can configure each machine to contact this particular machine to receive up-to-the-minute IP to name mappings. This is done by creating an `/etc/resolv.conf` file on every machine that looks something like this:

```
domain gentoo.org
nameserver 192.168.1.1
nameserver 192.168.1.2
```

In the above `/etc/resolv.conf`, I tell the system that any host names that are not qualified (such as "testbox" as opposed to "testbox.gentoo.org," etc.) should be considered to be *local* hostnames. I also specify that I have a DNS server running on 192.168.1.1, as well as a backup one running on 192.168.1.2. Actually, nearly all network-connected Linux PCs already have a nameserver specified in their `resolv.conf` file, even if they aren't on a LAN. This is because they are configured to use a DNS server at their Internet Service Provider, in order to map hostnames to IP addresses (so that users on that system can do things like browse the Web and head over to well-known sites like `ibm.com` without having to refer to them by IP address!).

Connecting to the outside

Speaking of connecting to the Internet, how would we configure our simple 3-system LAN so that it connect to systems on the "outside?" Typically, we'd purchase a router of some kind that can connect to both our Ethernet network *and* a DSL or Cable modem, or a T1 or phone line. This router would be configured with an IP address so that it could communicate with the systems on our LAN. In turn, we would configure every system on our LAN to use this router as its *default route*, or gateway. What this means is that any network data addressed to a system that *isn't* on our LAN would be routed to our router, which would take care of forwarding to remote systems outside our LAN. Generally, your distribution's system initialization scripts handle the setting of a default route for you. The command they use to do this probably looks something like this:

```
route add -net default gw 192.168.1.80 netmask 0.0.0.0 metric 1
```

In the above `route` command, the default route is set to 192168.1.80 -- the IP address of the router. To view all the routes configured on your system, you can type `route -n`. The route with a destination of "0.0.0.0" is the default route.

Homework

So far, we've given you a very brief introduction to Linux networking concepts. Unfortunately, we simply don't have the room to cover everything you need to know, such as how to select appropriate IP addresses, network masks, broadcast addresses, etc. In fact, there's quite a bit more information you'll need to learn in order to prepare for the LPI Level 102 exam.

Fortunately, the topic of Linux networking is one of the most comprehensively documented Linux topics around. In particular, we recommend that you read the [Linux Network Administrators Guide](#), available from [Linuxdoc.org's "Guides" section](http://Linuxdoc.org's \), especially sections 2 through 6. Accompanied with our gentle introduction to Linux networking, the Linux Network Administrators Guide should get you up to speed in no time!

Section 3. Internet services

Introducing inetd

A single Linux system can provide dozens, even hundreds, of network services. For example, when you use the telnet program, you are accessing the telnet service on a remote system. Likewise, when you use the ftp program, you are connecting to the ftp service on the remote system.

In order to provide these services, the remote system either runs an instance of each server to accept connections (for example `/usr/sbin/in.telnetd` and `/usr/sbin/in.ftpd`), or runs `inetd`. The `inetd` program accepts each incoming connection and starts the appropriate services to handle the connection based on its type. For this reason, `inetd` is also known as the "Internet superserver."

On a typical Linux installation, `inetd` handles most incoming connections. Only a few programs (such as `sshd` and `lpd`) handle their own network communication without relying on `inetd` to accept incoming connections.

Configuring inetd: /etc/services

The previous panel mentioned that `inetd` classifies incoming connections based on type. Each incoming connection includes some identification fields in the TCP/IP header. The fields that interest us the most are source address, destination address, protocol, and port number. Incoming connections are classified by `inetd` based on port number and protocol (usually TCP or UDP, see `/etc/protocols` for the complete list of services that can be serviced by `inetd`).

Each line has the format:

```
service-name port-number/protocol-name aliases # comment
```

For example, let's investigate the top few entries:

```
# grep ^[^\#] /etc/services | head -5
tcpmux    1/tcp    # TCP port service multiplexer
echo      7/tcp
echo      7/udp
discard   9/tcp    sink null
discard   9/udp    sink null
```

In general, `/etc/services` already contains all the useful service names and ports. If you wish to add your own, you should first consult the list of [assigned port numbers](#).

Configuring inetd; /etc/inetd.conf

The actual configuration of `inetd` is done in `/etc/inetd.conf`, which has the following format:

```
service-name socket-type protocol wait-flag user server-program
```

Since services are specified in `inetd.conf` by *service name* rather than port, they must be listed in `/etc/services` in order to be eligible for handling by `inetd`.

Let's look at some common lines from `/etc/inetd.conf`. For example, the telnet and ftp services:

```
# grep ^telnet /etc/inetd.conf
telnet stream tcp nowait root /usr/sbin/in.telnetd

# grep ^ftp /etc/inetd.conf
ftp stream tcp nowait root /usr/sbin/in.ftpd -l -a
```

For both of these services, the configuration is to use the TCP protocol, and run the server (`in.telnetd` or `in.ftpd`) as the root user. For a complete explanation of the fields in `/etc/inetd.conf`, see the *inetd(8)* man page.

Disabling services

Disabling a service in `inetd` is simple: Just comment out the line in `/etc/inetd.conf`. You probably don't want to remove the line entirely, just in case you need to reference it later. For example, some system administrators prefer to disable telnet for security reasons (since the connection is entirely cleartext):

```
# vi /etc/inetd.conf
[comment out undesired line]

# grep ^.telnet /etc/inetd.conf
#telnet stream tcp nowait root /usr/sbin/in.telnetd
```

Stopping/starting inetd using an init-script

The changes to `/etc/inetd.conf` that we made in the previous panel won't take effect until we restart the `inetd` program. Most distributions have an init-script in `/etc/init.d` or in `/etc/rc.d/init.d`:

```
# /etc/rc.d/init.d/inet stop
Stopping INET services:          [ OK ]

# /etc/rc.d/init.d/inet start
Starting INET services:        [ OK ]
```

In fact, you can usually use "restart" as a shortcut:

```
# /etc/rc.d/init.d/inet restart
Stopping INET services:        [ OK ]
Starting INET services:        [ OK ]
```

Stopping/starting inetd manually

If the helper script in the previous panel doesn't work for you, the old-fashioned method is even easier. You can stop `inetd` using the `killall` command:

```
# killall inetd
```

You can start it again simply by invoking it on the command line. It will automatically run in the background:

```
# /usr/sbin/inetd
```

And there's a shortcut to instruct `inetd` to reread the configuration file without actually stopping it: just send it the HUP signal:

```
# killall -HUP inetd
```

At this point you shouldn't be able to telnet or ftp into this system, since telnet and ftp are disabled. Try `telnet localhost` to check. If you need telnet or ftp access, all you need to do is to re-enable it!

Here's what happens for me:

```
# telnet localhost
telnet: Unable to connect to remote host: Connection refused
```

Introducing TCP wrappers

The `tcp_wrappers` package provides a tiny daemon called `tcpd` that is called by `inetd` instead of by the actual service daemon. The `tcpd` program logs the source address of each incoming connection, and can also filter them to allow connections only from trusted systems.

To use `tcpd`, you can insert it into your `inetd` as follows:

```
ftp stream tcp nowait root /usr/sbin/tcpd in.ftpd -l -a
telnet stream tcp nowait root /usr/sbin/tcpd in.telnetd
```

Logging with TCP wrappers

By default, connections are unrestricted but are logged. For example, we can restart `inetd` so that the changes from the previous panel take effect. Then some quick investigation should show the logged connections:

```
# telnet localhost
login: (press <ctrl-d> to abort)
```

```
# tail -1 /var/log/secure
Feb 12 23:33:05 firewall in.telnetd[440]: connect from 127.0.0.1
```

The telnet attempt was logged by tcpd, so it looks like we have things working. Since tcpd provides a consistent connection logging service, that frees up the individual service daemons from each needing to log connections on their own. In fact, it's similar to inetd doing the work of accepting connections, since that frees up each of the individual daemons from needing to accept their own connections. Isn't the simplicity of Linux (UNIX) marvelous?

Restricting access to local users with TCP wrappers

The tcpd program is configured using two files: `/etc/hosts.allow` and `/etc/hosts.deny`. These files have lines of the form:

```
daemon_list : client_list [ : shell_command ]
```

Access is granted or denied in the following order. The search stops at the first match:

- Access is granted when a match is found in **`/etc/hosts.allow`**
- Access is denied when a match is found in **`/etc/hosts.deny`**
- Access is granted if nothing matches

For example, to allow telnet access only to our internal network, we start by setting policy (reject all connections with a source other than localhost) in `/etc/hosts.deny`:

```
in.telnetd: ALL EXCEPT LOCAL
```

Restricting access to known hosts with TCP wrappers

There's no need to reload inetd, since tcpd is invoked each time there's a connection on the telnet port. So we can try it immediately:

```
# telnet box.yourdomain.com
Trying 10.0.0.1...
Connected to box.yourdomain.com.
Escape character is '^]'.
Connection closed by foreign host.
```

Slap! Rejected! (This is one of the few times in life that rejection is indicative of success.) To re-enable access from our own network, we insert the exception in `/etc/hosts.allow`:

```
in.telnetd: .yourdomain.com
```

At this point we're able to successfully telnet into our system again. This is just scraping the surface of the capabilities of tcp_wrappers. There's lots more information on tcp_wrappers in the [tcpd\(8\)](#) and [hosts_access\(5\)](#) man pages.

xinetd: inetd extended

Although inetd is the classic Internet superserver, there have been recent rewrites that attempt to add features and more security. The xinetd program replaces inetd in many modern distributions, including Red Hat and Debian. Some of its extended features are:

- Access control (built-in TCP wrappers)
- Extensive logging (connection durations, failed connections, etc.)
- Redirection of services from another host
- IPv6 support
- Configuration via snippets rather than one consolidated file

xinetd configuration

The configuration file for xinetd is `/etc/xinetd.conf`. Most often, that file contains just a few lines that set default configuration parameters for the rest of the services:

```
# cat /etc/xinetd.conf
defaults
{
    instances      = 60
    log_type       = SYSLOG authpriv
    log_on_success = HOST PID
    log_on_failure = HOST RECORD
}
includedir /etc/xinetd.d
```

The last line in that file instructs xinetd to read additional configuration from file snippets in the `/etc/xinetd.d` directory. Let's take a quick glance at the telnet snippet:

```
# cat /etc/xinetd.d/telnet
service telnet
{
    flags          = REUSE
    socket_type    = stream
    wait          = no
    user          = root
    server        = /usr/sbin/in.telnetd
    log_on_failure += USERID
}
```

As you can see, it's not hard to configure xinetd, and it's more intuitive than inetd. You can get lots more information about xinetd in the [xinetd\(8\)](#), [xinetd.conf\(5\)](#), and [xinetd.log\(5\)](#) man pages.

There's also lots of information on the Web regarding inetd, tcp_wrappers, and xinetd. Be sure to check out some of the links we've provided for these tools in the last section of this tutorial, Resources; they will give you a much better feel for the capability and configuration of these tools.

Section 4. Security overview

Introduction

Maintaining a completely secure system is impossible. With diligence, however, it is possible to keep your Linux machines secure enough that most casual crackers, script-kiddies, and other Bad Guys will be thwarted and go bug someone else. Remember that simply following this tutorial will not result in a secure system. Instead, we hope to touch on many of the major topic areas and gives you some useful examples of how to get started.

Linux system security can be divided into two parts: internal security and external security. *Internal security* refers to guarding against users accidentally or maliciously disrupting the system. *External security* refers to preventing unauthorized users from gaining access to the system.

This section will cover internal security first, then external security, and we'll finish with some general guidelines and tips.

File permissions for log files

Internal security can be a large task, depending on how much you are able to trust your users. The guidelines presented here are designed to prevent the casual user from accessing sensitive information and from unfairly using system resources.

Regarding file permissions, you may want to modify permissions for the following three cases:

First, log files in `/var/log` need not be world-readable. There is no reason for anybody other than root to be snooping in the logs. See [Part 4 of the LPI 101 series](#) for more information on syslog, plus the `logrotate(8)` man page for information on configuring that program to create logs with appropriate permissions.

File permissions for root's other files

Second, root's dot-files shouldn't be readable by an ordinary user. Check the files in root's home directory (`ls -la`) to make sure they're appropriately protected. You can even make the whole directory readable only by root:

```
# cd
# pwd
/root
# chmod 700 .
```

File permissions for user files

Finally, user files are often created world-readable by default. That probably isn't the expectation of your users, and it certainly isn't the best policy. You should set the default `umask` in `/etc/profile` using something like the following:

```
if [ "$UID" = 0 ]; then
    # root user; set world-readable by default so that
    # installed files can be read by normal users.
    umask 022
else
    # make user files secure unless they explicitly open them
    # for reading by other users
    umask 077
fi
```

You should consult the `umask(2)` and `bash(1)` man pages for more information on setting the `umask`. Note that the `umask(2)` man page refers to the C function, but the information it contains applies to the `bash` command as well. See [Part 3 of the LPI 101 series](#) for additional details on `umask`.

Finding SUID/SGID programs

A malicious user seeking root access will always look for programs on the system that have the SUID or SGID bit set. As we discussed in [Part 3 of the LPI 101 series](#), these bits cause the program to always run as the user or group that owns the file. Sometimes this is required for proper functioning of the program. The problem is that *any* program may contain a bug that would allow the user to gain privileges if the program is used improperly.

You should consider each program carefully to determine if it needs to have its SUID or SGID bits on. There may be SUID/SGID programs on your system that you don't need at all.

To search for programs of this nature, use the `find` command. For example, we could start searching for SUID/SGID programs in the `/usr` directory:

```
# cd /usr
# find . -type f -perm +6000 -xdev -exec ls {} \;
-rwsr-sr-x 1 root root 593972 11-09 12:47 ./bin/gpg
-r-xr-sr-x 1 root man 38460 01-27 22:13 ./bin/man
-rwsr-xr-x 1 root root 15576 09-29 22:51 ./bin/rcp
-rwsr-xr-x 1 root root 8256 09-29 22:51 ./bin/rsh
-rwsr-xr-x 1 root root 29520 01-17 19:42 ./bin/chfn
-rwsr-xr-x 1 root root 27500 01-17 19:42 ./bin/chsh
-rwsr-xr-x 1 lp root 8812 01-15 23:21 ./bin/lppasswd
-rwsr-x--- 1 root cron 10476 01-15 22:16 ./bin/crontab
```

In this list, I've already found a candidate for closer inspection: `lppasswd` is part of the CUPS printing software distribution. Since I don't provide print services on my system, I might consider removing CUPS, which will also remove the `lppasswd` program. There may be no security-compromising bugs in `lppasswd`, but why take the chance on a program I'm not using? Similarly, *all* services that you don't use should be turned off. You can always enable them if and when you need them.

Setting user limits with ulimit

The `ulimit` command in `bash` provides a method for limiting the usage of resources by a given user. Once a limit is lowered, there is no way to raise the limit for the life of the process. Furthermore, the limit is inherited by all child processes. The effect is that you can call `ulimit`

in `/etc/profile`, and the limits will irrevocably apply to all users (assuming they're running `bash` or another shell that runs `/etc/profile` on login).

To retrieve the current limits, use `ulimit -a`:

```
# ulimit -a
core file size      (blocks, -c) 0
data seg size      (kbytes, -d) unlimited
file size          (blocks, -f) unlimited
max locked memory  (kbytes, -l) unlimited
max memory size    (kbytes, -m) unlimited
open files         (-n) 1024
pipe size          (512 bytes, -p) 8
stack size         (kbytes, -s) unlimited
cpu time           (seconds, -t) unlimited
max user processes (-u) 3071
virtual memory     (kbytes, -v) unlimited
```

It can be quite tricky to set these limits in a way that actually increases the security of your system without causing problems for legitimate users, so be careful when adjusting these settings.

Setting CPU time limits with ulimit

As an example of `ulimit`, let's try setting the CPU time for a process to 1 second, then make it timeout with a busy loop. Make sure to start a new `bash` process (as we do below) in which to try it; otherwise you'll be logged out!

```
# time bash
# ulimit -t 1
# while true; do true; done
Killed

real    0m28.941s
user    0m1.990s
sys     0m0.017s
```

In the example above, "user" time plus "sys" time equals total CPU time used by the process. When the `bash` process reached the 2-second mark, Linux judged that it had exceeded the 1-second limit, so the process was killed. Cool, eh?

Note: One second was just an example. Don't do this to your users! Even multiple hours is bad, since X can really rack up the time (my current session has used 69+ hours of CPU time). For a real implementation, you may want to `ulimit` something other than CPU time.

User limits, continued

You may also want to limit things such as the number of simultaneous logins or disk usage. These aren't covered by `ulimit`; instead you should look into one of the following packages:

- [Clobberd](#) monitors user activity, and meters resources such as time and network activity.
- [ldled](#) can log out users that have been idle for too long or who have been logged on for too

long. It can also prevent users from being logged in too many times, and refuse users from being logged in at all.

- [Part 4 of the LPI 101 series](#) discusses the implementation of filesystem quotas.

Intrusion prevention

External security can be split into two categories: intrusion prevention and intrusion detection. Intrusion prevention measures are taken to prevent unauthorized access to a system. If these measures fail, intrusion detection may prove useful in determining when unauthorized access has occurred, and what damage has been done.

A full Linux installation is a large and complex system. It's difficult to keep track of everything that's installed, and even harder to configure each package's security features. The problem becomes simpler when fewer packages are installed. A first step to intrusion prevention is to remove packages you don't need. Take a look back at [Part 4 of the LPI 101 series](#) for a review of packaging systems.

Turning off unused network services (superserver)

Turning off unused network services is always a good way to improve your intrusion prevention. For example, if you are running an Internet superserver (such as `inetd` or `xinetd` described earlier in this tutorial), then `in.rshd`, `in.rlogind`, and `in.telnetd` are often enabled by default. These network services have nearly all been superseded by more secure alternatives such as `ssh`.

To disable services in `inetd`, simply comment out the appropriate line in `/etc/inetd.conf` by prepending "`#`:" then restart `inetd`. (This was described previously in this tutorial, so glance back a few panels if you need a refresher.)

To disable services in `xinetd`, you can do something similar with the appropriate snippet in `/etc/xinetd.d`. For example, to disable `telnet`, either comment out the entire content of the file `/etc/xinetd.d/telnet`, or simply delete the file. Restart `xinetd` to complete the procedure.

If you're using `tcpd` in conjunction with `inetd`, or if you're using `xinetd`, you also have the option of limiting incoming connections to trusted hosts. For `tcpd`, see the earlier sections in this tutorial. For `xinetd`, search for "`only_from`" in the `xinetd.conf(5)` man page.

Turning off unused network services (standalone servers)

Some servers are not launched by `inetd` or `xinetd`, but are instead running all the time as "standalone" servers. This often includes servers such as `atd`, `lpd`, `sshd`, `nfsd`, and others. In fact, `inetd` and `xinetd` are both standalone servers themselves, and if you have commented out all of the services in their respective config files, you may choose to turn them off completely.

Standalone servers are usually started by the `init` system when the system boots up or changes runlevels. If you don't remember how runlevels work, take a look at [Part 4 of the LPI](#)

[101 series.](#)

To stop the init system from starting a server, find the symlinks to its startup script in each runlevel directory, and delete them. The runlevel directories are usually named `/etc/rc3.d` or `/etc/rc.d/rc3.d` (for runlevel 3). You'll also want to check the other runlevels.

Once the runlevel symlinks for the service are removed, you will still need to shut down the currently running server. It is best to do this with the service's init script, usually found in `/etc/init.d` or `/etc/rc.d/init.d`. For example, to shut down `sshd`:

```
# /etc/init.d/sshd stop
* Stopping sshd... [ ok ]
```

Testing your changes

After you've modified your `inetd` or `xinetd` configuration to disable or restrict services, or to shut down a server with its init script, you should test your changes. You can test tcp ports using the `telnet` client by specifying the service name or number. For example, to verify that `rlogin` has been disabled:

```
# grep ^login /etc/services
login      513/tcp
# telnet localhost 513
Trying 127.0.0.1...
telnet: Unable to connect to remote host: Connection refused
```

In addition to the standard `telnet` client, you should look into the possibility of using utilities for testing the "openness" of your system. We recommend `netcat` and `nmap`.

ncat is the network Swiss Army knife: it is a simple UNIX utility that reads and writes data across network connections, using TCP or UDP protocol. **nmap** is a utility for network exploration or security auditing. Specifically, `nmap` scans ports to determine what's open.

You'll find links to these utilities in the last section of this tutorial, [Resources](#).

Refusing logins for maintenance

In addition to the above methods, there is a generic way to refuse logins by creating the file `/etc/nologin`. Usually this method is used for short-term maintenance operations. Logins by root will still be allowed, but logins by other users will be denied. For example:

```
# cat > /etc/nologin
=====
System is currently undergoing maintenance
until 2:00. Please come back later.

=====
# telnet localhost
login: agriffis
```

```
Password:
=====
System is currently undergoing maintenance
until 2:00. Please come back later.
```

```
=====
```

```
Login incorrect
```

Be sure to delete the file when you're done with maintenance, otherwise nobody will be able to login until you remember! Not that I've ever done this, no, not me... ;-)

Introducing iptables (ipchains)

The `iptables` and `ipchains` commands are used to adjust and inspect the network packet filter rules in a running Linux kernel. The `ipchains` command was used for 2.2.x versions of the kernel, and although it can still be used with 2.4.x kernels, it has been superseded by `iptables`.

The packet filter rules can be set up to do both firewall and router activities. You can inspect your current rules with the `-L` option to `iptables`:

```
# iptables -L
Chain INPUT (policy ACCEPT)
target    prot opt source                destination

Chain FORWARD (policy ACCEPT)
target    prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target    prot opt source                destination
```

This is an example of a wide-open system, with no routing or firewalling enabled.

iptables and Linux packet filter

Using the Linux packet filter effectively requires a solid understanding of TCP/IP networking and how it is implemented in the Linux kernel. The netfilter home page (see [Resources](#), the last section of this tutorial, for a link) is a good place to learn more.

Until you're comfortable building your own ruleset, there are many scripts out there that can get you started, as long as you trust their authors. One of the most complete is **gShield** (see [Resources](#)). You can adjust its well-commented and fairly simple configuration file to set up most normal forms of packet filter rules.

Intrusion detection - syslogs

Intrusion detection is often neglected by system administrators who trust in the intrusion prevention devices they have in place. Unfortunately, this means that when a hacker finds a crack through which to crawl, the system may be under their control for a long period of time

before their presence is noticed.

The most basic form of intrusion detection is to pay attention to your syslogs. These usually appear in the `/var/log` directory, although the actual filenames will vary depending on your distribution and configuration.

```
# less /var/log/messages
Feb 17 21:21:38 [kernel] Vendor: SONY Model: CD-RW CRX140E Rev: 1.0n
Feb 17 21:21:39 [kernel] eth0: generic NE2100 found at 0xe800, Version 0x031243,
DMA 3 (autodetected), IRQ 11 (autodetected).
Feb 17 21:21:39 [kernel] ne.c:vl.10 9/23/94 Donald Becker (becker@scyld.com)
Feb 17 21:22:11 [kernel] NVRM: AGPGART: VIA MVP3 chipset
Feb 17 21:22:11 [kernel] NVRM: AGPGART: allocated 16 pages
Feb 17 22:20:05 [PAM_pwdb] authentication failure; (uid=1000)
-> root for su service
Feb 17 22:20:06 [su] pam_authenticate: Authentication failure
Feb 17 22:20:06 [su] - pts/3 chouser-root
```

It can take some practice to understand all these messages, but most of the important ones are fairly clear. For example, at the end of this log we can see that user "chouser" tried to use `su` to become root and failed.

Intrusion detection - tripwire

There are a number of packages available that can take a "snapshot" of your whole filesystem and compare it to earlier snapshots to see what has changed. If you can clearly define which files *should* change as part of the normal operation of your system, these packages can very quickly alert you to the presence and activity of a hacker.

Tripwire is one of the most popular of these intrusion detection packages (see Resources at the end of this tutorial for a link). Once you have installed tripwire, you must customize its configuration file so that it knows which files should change and which should not. You will also need to tell it how to send you reports of what has changed, and how often it should run (usually once per day).

Intrusion detection - portsentry

The **PortSentry** package from Psionic Technologies is actually a bit of a cross between intrusion prevention and detection. It watches your network connection, and if it sees any attempts to connect to your system that it deems "suspicious," it will log the event and then block it from happening again. It, too, can be found in Resources at the end of this tutorial.

When you have it installed and running, you will be able to see any attempted connections and how PortSentry responded to them in your syslog:

```
# tail /var/log/messages
Oct 15 00:21:24 mycroft portsentry[603]: attackalert:
  SYN/Normal scan from host: 302.174.40.34/302.174.40.34 to TCP port: 111
Oct 15 00:21:24 mycroft portsentry[603]: attackalert:
  Host 302.174.40.34 has been blocked via wrappers with string:
  "ALL: 302.174.40.34"
```

```
Oct 15 00:21:24 mycroft portsentry[603]: attackalert:
  Host 302.174.40.34 has been blocked via dropped route using command:
  "/sbin/route add -host 302.174.40.34 reject"
Oct 15 00:21:24 mycroft portsentry[603]: attackalert:
  SYN/Normal scan from host: 302.174.40.34/302.174.40.34 to TCP port: 111
Oct 15 00:21:24 mycroft portsentry[603]: attackalert:
  Host: 302.174.40.34/302.174.40.34 is already blocked Ignoring
Oct 15 00:33:59 mycroft portsentry[603]: attackalert:
  SYN/Normal scan from host: 302.106.103.19/302.106.103.19 to TCP port: 111
Oct 15 00:33:59 mycroft portsentry[603]: attackalert:
  Host 302.106.103.19 has been blocked via wrappers with string:
  "ALL: 302.106.103.19"
Oct 15 00:33:59 mycroft portsentry[603]: attackalert:
  Host 302.106.103.19 has been blocked via dropped route using command:
  "/sbin/route add -host 302.106.103.19 reject"
```

General guidelines: keeping software up to date

Since all software has the potential for security holes, it's important to install security fixes for packages as soon as they become available. This is the single most often offered piece of advice by security experts, and the single most often ignored piece of advice by novice administrators. Don't learn this lesson the hard way -- when your boxes have been rooted by a years-old backdoor because you neglected to keep patches up to date.

The debate over whether open source or closed-source software is more secure is a hotly debated one. The best conclusion that can be drawn to date is that they are both usually adequate *when properly administered*, which includes keeping security patches up to date!

Several Web sites can help you keep your software up to date, and keep you aware of known threats. These include the security-conscious [CERT](#) and [SecurityFocus' BugTraq list](#), as well as your normal software update sites like [freshmeat.net](#) and your distribution's home page. We'll repeat these URLs also in Resources, but security is such an important issue that -- if you are not already familiar with these sites, we do recommend that you take a few minutes to visit the first two now.

General guidelines: high-quality passwords

As mundane as it may sound, choosing high-quality passwords for yourself, and "encouraging" (that is, mandating that) your users to do the same, is a cornerstone of good security. Remember to avoid common words and names, especially any that are related to you such as the name of a friend, where you work, or your pet's name. Also avoid guessable numbers such as your birthday or anniversary. Instead try to use random combinations of letters, numbers, and punctuation.

General guidelines: testing your security

Testing the security of your system is important, but don't let a successful test lull you into a false sense of security. Just because these testing tools don't find a hole is no guarantee that someone with knowledge and imagination -- and a whole lot of time on their hands -- will also fail.

We already mentioned nmap and netcat for testing network security. It is also a good idea to check for weak passwords, especially if your system has multiple users. There are many tools available, such as the ones we've put into Resources at the end of this tutorial.

Section 5. Printing

Introduction

This section will walk you through the set-up and use of the classic UNIX printing system on Linux, sometimes called Berkeley LPD. There are other systems available for Linux that are not covered here; see the Resources section at the end of the tutorial for information on these.

Physically installing a printer is beyond the scope of this tutorial. Once it's correctly hooked up, you'll want to install a print spooler daemon so that machines on the network (including the one housing the spooler) can send print jobs to the printer.

Installing a print spooler daemon (lpd)

One of the best LPD print spoolers is [LPRng](#). The method of installation depends on your distribution; see [Part 1 of the LPI 102 series](#) for details on installing software packages in either Red Hat or Debian.

Once it's installed, the print spooler daemon (officially the Line Printer Daemon) can be run from the command line. Log in as a normal user and try:

```
$ /usr/sbin/lpd --help
--X option form illegal
usage: lpd [-FV] [-D dbg] [-L log]
Options
-D dbg      - set debug level and flags
              Example: -D10,remote=5
              set debug level to 10, remote flag = 5
-F          - run in foreground, log to STDERR
              Example: -D10,remote=5
-L logfile  - append log information to logfile
-V          - show version info
```

Now that the daemon is installed, you should make sure that it's set up to run automatically. Your distribution's LPRng package may have set this up for you already, but if not, see [Part 4 of the LPI 101 series](#) for information on using runlevels to start daemons such as lpd automatically.

Basic printer settings (/etc/printcap)

The print spooler daemon acts as a sort of pipeline. It accepts print jobs coming in from various print clients, and then passes these jobs along to the appropriate printer. While the printer is busy, these jobs "spool," waiting for their chance to get printed.

When printing on the local printer, both of ends of this "pipeline" are described by the configuration file `/etc/printcap` (sometimes located at `/etc/lprng/printcap`). Each entry in the printcap (which is short for *printer capabilities*) describes one print spool:

```
$ more /etc/printcap
```

```
lp|Generic dot-matrix printer entry:\
:lp=/dev/lp0:\
:sd=/var/spool/lpd/lp:\
:pl#66:\
:pw#80:\
:pc#150:\
:mx#0:\
:sh:
```

Note that the last line of the entry does **not** have a trailing backslash (\).

Your distribution may have other entries, and they may be more complex, but they should all have roughly this form. The name of this entry comes first, **lp**, followed by a longer description of this spool. The keyword/value pair **lp=/dev/lp0** specifies the Linux device were print jobs in this spool will be printed, and the **sd** keyword gives the directory where jobs will be held until they can be printed.

The rest of the keyword/value pairs provide details about what type of printer is hooked up to /dev/lp0. They are described in the *printcap* man page, and we'll cover some of them a little bit later.

Creating spool directories

If you create an entry, you'll need to make sure the spool directory exists and has the correct permissions. You want the printer daemon, which is usually run as user **lp**, to have access to the spool directory. You'll have to run these commands as root:

```
# mkdir -p /var/spool/lpd/lp
# chown lp /var/spool/lpd/lp
# chmod 700 /var/spool/lpd/lp
# checkpc -f
# /etc/init.d/lprng restart
```

LPRng includes a helpful tool for checking your printcap. It will even set up the spool directory for you, if you forget to do it manually:

```
# checkpc -f
```

Finally, restart the lpd. You'll need to do this any time you change the printcap, in order for your changes to take effect. You may need to use lpd instead of lprng:

```
# /etc/init.d/lprng restart
```

The older Berkeley printing system doesn't include a checkpc tool, so you'll have to try printing pages to your various printers yourself to make sure the printcap and spool directories are correct.

Using print spooler clients

The print spooler comes with several clients for communicating with the server daemon. The one you're likely to use most often is **lpr**, which simply sends a file to the server to be queued

up in a print spool and then printed. To try it out, first find or make a small sample text file. Then:

```
$ lpr sample.txt
```

If it worked, you shouldn't see any response on the screen, but your printer should start going, and soon you'll have a hard copy of your sample text. Don't worry if it doesn't come out looking quite right; we'll set up filters a bit later that should ensure that all sorts of file formats print correctly.

You can examine the list of print jobs in the print spool queue with the **lpq** command. The **-P** option specifies the name of the queue to display; if you leave it off, lpq will use the default spool, just like lpr did before:

```
$ lpq -Plp
Printer: lp@localhost 'Generic dot-matrix printer entry'
Queue: 1 printable job
Server: pid 1671 active
Unspooler: pid 1672 active
Rank   Owner/ID                Class Job Files      Size Time
active chouser@localhost+670  A    670 sample.txt      8 21:57:30
```

If you want to stop a job from printing, use the **lprm** command. You might want to do this if a job is taking too long, or if a user accidentally sends the same file more than once. Just copy the job id from the lpq listing above:

```
$ lprm chouser@localhost+670
Printer lp@localhost:
  checking perms 'chouser@localhost+670'
  dequeued 'chouser@localhost+670'
```

You can do many other operations on a print spool using the interactive tool **lpc**. See the *lpc* man page for details.

Printing to a remote LPD server

Even if you don't have a printer on your local machine, you can still use lpd to send a print job across the network to a printer that is attached to some other machine. On the client machine, you can add an entry to your /etc/printcap that looks like a local printer but actually routes print jobs to the server machine. This entry will look something like this:

```
farawaylp|Remote printer entry:\
:rm=faraway:\
:rp=lp:\
:sd=/var/spool/lpd/farawaylp:\
:mx#0:\
:sh:
```

Here the name of the machine where we want to print is **faraway**, and the name of the printer on *that* machine is **lp**. The spool directory, /var/spool/lpd/farawaylp, is where print jobs will be held locally until they can be sent to the remote print spooler, where they may be spooled again before being sent to the printer. Again, you will need to create this spool

directory and set its permissions:

```
# mkdir -p /var/spool/lpd/farawaylp
# chown lp /var/spool/lpd/farawaylp
# chmod 700 /var/spool/lpd/farawaylp
# checkpc -f
# /etc/init.d/lprng restart
```

Locally, we have given this remote printer the name **farawaylp**, so we can send print jobs to it thusly:

```
$ lpr -Pfarawaylp sample.txt
```

Printing to a remote MS Windows or Samba server

Thanks to Samba, printing to a remote Microsoft Windows print server is only slightly more complicated. First, add the local printcap entry:

```
smb|Remote windows printer:\
:if=/usr/bin/smbprint:\
:lp=/dev/null:\
:sd=/var/spool/lpd/smb:\
:mx#0:
```

The new key here is **if**, the input filter. Pointing this to the smbprint script will cause the print job to be sent to a Windows server instead of the lp device. We still have to list a device (/dev/null in this case) which the print daemon uses for locking. But no print jobs will actually be sent there.

Don't forget to create the spool directory!

```
# mkdir -p /var/spool/lpd/smb
# chown lp /var/spool/lpd/smb
# chmod 700 /var/spool/lpd/smb
# checkpc -f
# /etc/init.d/lprng restart
```

In your favorite editor, create a .config file in the spool directory named above. In this case, /var/spool/lpd/smb/.config:

```
server="WindowsServerName"
service="PrinterName"
password=""
user=""
```

Adjust these values to point to the Windows machine and printer name that you want to use, and you're done:

```
$ lpr -P smb sample.txt
```

The smbprint script should come with Samba, but it isn't included in all distributions. If you

can't find it on your system, you can get it from the [Samba HOWTO](#).

Magicfilter

So far we've only tried printing text files, which isn't terribly exciting. Generally any one printer can only print one format of graphics file -- and yet there are dozens of different formats that we would like to print: PostScript, gif, jpeg, and many more. A program named *Magicfilter* acts as an input filter, much like smbprint does. It doesn't convert the file formats, rather it provides a framework for identifying the type of document you're trying to print, and runs it through an appropriate conversion tool: conversion tools must be installed separately. By far, the most important of these is *Ghostscript*, which can convert files from Postscript format to the native format of many printers.

Adjusting printcap to point to Magicfilter

Once these tools are installed, you simply need to adjust your printcap one more time. Add an **if** key to point to the Magicfilter that goes with your printer:

```
lp|The EPSON Stylus Color 777 sitting under my desk:\
:if=/usr/share/magicfilter/StylusColor-777@720dpi-filter:\
:gqfilter:\
:lp=/dev/usb/lp0:\
:sd=/var/spool/lpd/lp:\
:pl#66:\
:pw#80:\
:pc#150:\
:mx#0:\
:sh:
```

There are filters for dozens and dozens of different printers and printer settings in /usr/share/magicfilter, so be sure you use the right one for your printer. Each of these filters is a text file, and usually the full name of the printer is at the top. This may help you if the file name of the filter isn't clear.

I also added a gqfilter flag to this printcap entry, which will cause the input filter to be used even when the print job is coming from a remote client. This only works with LPRng.

Since I set up the /var/spool/lpd/lp print spool directory earlier, I only need to check my printcap syntax and restart the server:

```
# checkpc -f
# /etc/init.d/lprng restart
```

Now you are able to print all sorts of documents, including Postscript files. In other words, choosing "Print" from a menu in your favorite web browser should now work.

Apsfilter as alternative to Magicfilter

Apsfilter provides many of the features of Magicfilter, but it also helps you set up your spool

directories, printcap entries, and so on. You will still need to make sure you have Ghostscript installed, but then you can follow the very complete instructions provided in the [Apsfilter handbook](#).

Section 6. Resources and feedback

Resources

You can learn more about **configuring inetd and xinetd** from the article [Configuring inetd.conf securely](#) and from the [xinetd home page](#). When adding your own service names and ports to /etc/services, remember to check first that they don't conflict with the [assigned port numbers](#).

The [netfilter home page](#) is a good place to start learning more about **iptables and the Linux packet filter**. Until you're comfortable building your own ruleset, you might want to use an existing script for this. We recommend [gShield](#).

Useful security tools include [Tripwire](#), one of the most popular intrusion detection packages, and [Psionic Technologies' PortSentry](#), which is a cross between intrusion prevention and detection. (The [LinuxWorld](#) article [How to stop crackers with PortSentry](#) offers advice on installation and configuration.) Finally, be sure to become familiar with Wietse Venema's [TCP Wrappers](#), which allow monitoring of and control over connections to your system. You can view the [TCP wrappers README](#) online (it's also available in /usr/share/doc/tcp_wrappers-7.6). Authenticating users can be much easier with [Pluggable Authentication Modules](#) (also known as PAM).

Is your network wide open? Consider trying these two utilities for checking the vulnerability (or "openness") of your system: [netcat](#) is a simple UNIX utility that reads and writes data across network connections, using TCP or UDP protocol; [nmap](#) is a utility for network exploration or security auditing. Specifically, nmap scans ports to determine what's open.

Password checkers and other tools that will test how easy it is to guess your passwords (and those of your users) include [John the Ripper](#) from the [Openwall Project](#), built for just this purpose. You may also want to try a comprehensive checker like [SAINT](#).

These **security sites** should be among the most-visited by any systems administrator: [CERT](#) is a federally-funded center operated by Carnegie Mellon University. They study Internet security and vulnerabilities, publish security alerts, and research other security issues. [BugTraq](#), hosted by Security Focus, is a full-disclosure moderated mailing list for the detailed discussion and announcement of computer security vulnerabilities. Even if you aren't particularly interested in this aspect of administration, a subscription to this list can be quite valuable, as simply scanning subject lines may alert you to vulnerabilities on your own systems that you might otherwise discover much later, or not at all.

Some **more security sites** we recommend highly for getting a better grip on the security of your Linux machines are the [Linux Security HOWTO](#), [O'Reilly's Security Page](#), and of course developerWorks' [Security zone](#) (although it has a greater emphasis on secure **programming** practices, it does also feature administrative security).

Printing and spooling resources you'll want to check out include the [LPRng print spooler home page](#) and the [Spooling Software overview](#) from the Printing HOWTO. Of course, the [Printing HOWTO](#) itself is a valuable resource, as is [LinuxPrinting.org](#).

For help with **specific printers**, consult the [Serial HOWTO](#). Also the [USB guide](#) offers valuable information on (you guessed it) USB printers.

Samba is a great help in heterogeneous networks. When setting up printing for this kind of environment, you will want to check out the [Samba home page](#) as well as the [Samba HOWTO](#), with good printer sharing details.

The two **printer filters** we discussed were [Magicfilter](#) and [Apsfilter](#). Remember that both need a conversion program (we recommend [Ghostscript](#)), as they do not do conversion themselves. If you are using the latter filter, you may also find the [Apsfilter handbook](#) to be quite useful.

In addition, we recommend the following general resources for learning more about Linux and preparing for LPI certification in particular:

You'll find a wealth of guides, HOWTOs, FAQs, and man pages at <http://www.linuxdoc.org>. Be sure to check out [Linux Gazette](#) and [LinuxFocus](#) as well.

The Linux Network Administrator's guide, available from [Linuxdoc.org's "Guides" section](#), is a good complement to this series of tutorials -- give it a read! You may also find Eric S. Raymond's [Unix and Internet Fundamentals HOWTO](#) to be helpful.

In the *Bash by example* article series on *developerWorks*, learn how to use `bash` programming constructs to write your own `bash` scripts. This series (particularly parts 1 and 2) are excellent additional preparation for the LPI exam:

- [Bash by example, Part 1: Fundamental programming in the Bourne-again shell](#)
- [Bash by example, Part 2: More bash programming fundamentals](#)
- [Bash by example, Part 3: Exploring the ebuild system](#)

The [Technical FAQ for Linux Users](#) by Mark Chapman is a 50-page in-depth list of frequently-asked Linux questions, along with detailed answers. The FAQ itself is in PDF (Acrobat) format. If you're a beginning or intermediate Linux user, you really owe it to yourself to check this FAQ out. The [Linux glossary for Linux users](#), also from Mark, is also excellent.

If you're not too familiar with the `vi` editor, you should check out Daniel's [tutorial on Vi](#). This *developerWorks* tutorial will give you a gentle yet fast-paced introduction to this powerful text editor. Consider this must-read material if you don't know how to use `vi`.

For more information on the Linux Professional Institute, visit the [LPI home page](#).

Your feedback

We look forward to getting your feedback on this tutorial. Additionally, you are welcome to contact the lead author, Daniel Robbins, directly at [drobbins@gentoo.org](mailto:d Robbins@gentoo.org).

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.