LPI exam prep: Networking configuration

Intermediate Level Administration (LPIC-2) topic 205

Skill Level: Intermediate

David Mertz, Ph.D. (mertz@gnosis.cx)
Developer
Gnosis Software

08 Nov 2005

This is the first of seven tutorials covering intermediate network administration on Linux®. In this tutorial, David Mertz shows you how to configure a basic TCP/IP network, from the hardware layer (usually Ethernet, modem, ISDN, or 802.11), through the routing of network addresses. Higher level servers that may operate on these configured networks are covered in later tutorials.

Section 1. Before you start

Learn what these tutorials can teach you and how you can get the most from them.

About this series

The Linux Professional Institute (LPI) certifies Linux system administrators at two levels: *junior level* (also called "certification level 1") and *intermediate level* (also called "certification level 2"). To attain certification level 1, you must pass exams 101 and 102; to attain certification level 2, you must pass exams 201 and 202.

developerWorks offers tutorials to help you prepare for each of the four exams. Each exam covers several topics, and each topic has a corresponding self-study tutorial on developerWorks. For LPI exam 202, the seven topics and corresponding developerWorks tutorials are:

Table 1. LPI exam 202: Tutorials and topics

LPI exam 202 topic	developerWorks tutorial	Tutorial summary
Topic 205	LPI exam 202 prep (topic	(This tutorial) Learn how to

	205): Networking configuration	configure a basic TCP/IP network, from the hardware layer (usually Ethernet, modem, ISDN, or 802.11), through the routing of network addresses. See detailed objectives below.
Topic 206	LPI exam 202 prep (topic 206): Mail and news	Coming soon
Topic 207	LPI exam 202 prep (topic 207): DNS	Coming soon
Topic 208	LPI exam 202 prep (topic 208): Web services	Coming soon
Topic 210	LPI exam 202 prep (topic 210): Network client management	Coming soon
Topic 212	LPI exam 202 prep (topic 212): System security	Coming soon
Topic 214	LPI exam 202 prep (topic 214): Network troubleshooting	Coming soon

To start preparing for certification level 1, see the developerWorks tutorials for LPI exam 101. To prepare for the other exam in certification level 2, see the developerWorks tutorials for LPI exam 201. Read more about the entire set of developerWorks LPI tutorials.

The Linux Professional Institute does not endorse any third-party exam preparation material or techniques in particular. For details, please contact info@lpi.org.

About this tutorial

Welcome to "Networking configuration", the first of seven tutorials covering intermediate network administration on Linux. In this tutorial, you learn how to configure a basic TCP/IP network from the hardware layer (usually Ethernet, modem, ISDN, or 802.11) through the routing of network addresses. Higher level servers that may operate on these configured networks are covered in later tutorials.

is organized according to the LPI objectives for this topic. Very roughly, expect more questions on the exam for objectives with higher weight.

Table 2. Networking configuration: Exam objectives covered in this tutorial

LPI exam objective	Objective weight	Objective summary
2.205.1	Weight 5	Configure a network device to

Basic networking configuration		be able to connect to a local network and a wide-area network. This objective includes being able to communicate between various subnets within a single network, configure dialup access using mgetty, configure dialup access using a modem or ISDN, configure authentication protocols such as PAP and CHAP, and configure TCP/IP logging.
2.205.2 Advanced network configuration and troubleshooting	Weight 3	Configure a network device to implement various network authentication schemes. This objective includes configuring a multi-homed network device, configuring a virtual private network, and resolving networking and communication problems.

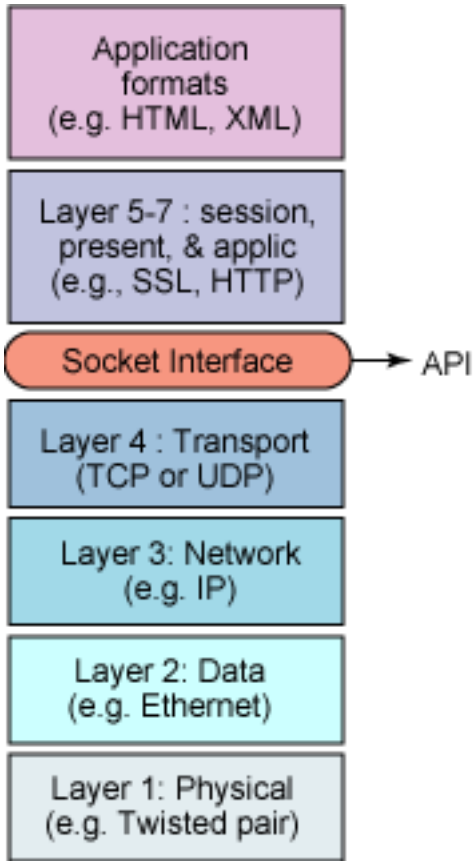
Prerequisites

To get the most from this tutorial, you should have a basic knowledge of Linux and a working Linux system on which you can practice the commands covered in this tutorial.

Section 2. About network configuration

When thinking about Linux networking and network configuration, it helps to keep in mind the *OSI seven-layer reference model*:

Figure 1. The OSI seven-layer reference model



What we call "network configuration" essentially lives on the second and third layers -- the *Data Link Layer* and the *Network Layer* -- and in the interfaces between them. In practice, this amounts to either Ethernet or serial interfaces like modems for the Data Link Layer, and the Internet Protocol (IP) for the Network Layer. Later tutorials in this series deal with higher-level layers, though most server applications discussed do not cleanly separate all seven layers (or even the top four where they operate).

The first network layer is the *Physical Layer*, the actual wires (or wireless channels) and circuits. A practical network administrator needs to be ready to inspect cabling and install new network peripherals from time to time (but those issues are outside the scope of these tutorials). Clearly, however, a loose wire, fried Ethernet card, or broken plug is just as capable of creating network problems as is misconfigured software.

Layer four is the *Transport Layer*; concretely, this means either TCP or UDP in IP networks. TCP and UDP are used at higher levels via the Berkeley Sockets Interface, a well-tested library found on all modern computer systems. For background on how applications (such as those discussed in later tutorials of this series) use TCP or UDP read the tutorials "Programming Linux sockets, Part 1" and "Programming Linux sockets, Part 2."

Other resources

As with most Linux tools, the manpages contain valuable information. For more in-depth information, the Linux Documentation Project has a variety of useful documents, especially its HOWTOs. Many books on Linux networking, such as O'Reilly's *TCP/IP Network Administration* by Craig Hunt, can be quite helpful. You'll find links to these and more resources in this tutorial's Resources section.

Section 3. Basic networking configuration

Address resolution protocol

The first thing to understand about Ethernet devices -- either 802.11a/b/g wireless or the more traditional CAT5/CAT6 wired networks -- is that every Ethernet device has a unique six-byte ID in it. Those IDs are assigned in blocks to manufacturers; you can look up the Ether number assignments at IANA. Ethernet generally "just works" at the hardware level, but a system needs to map an Ethernet ID to the IP address it will use to enable IP networking.

The Address Resolution Protocol (ARP) lets machines discover each others' IP address within a local Ethernet network. As a protocol, ARP is generally implemented within network device drivers (kernel modules); the arp tool lets you examine the status of the ARP system and tweak it in some ways. At this point, we assume that each machine has been configured to know its own IP address, either by static assignment or dynamically with DHCP.

When a Linux system (or any device with Ethernet) wishes to address an IP address, the ARP request message "who is X.X.X.X tell Y.Y.Y.Y" is sent using the Ethernet broadcast address. The target system forms an ARP response "X.X.X.X is hh:hh:hh:hh:hh:hh" and sends it to the requesting host. An ARP response is cached for a short time in /proc/net/arp to avoid the need to continually reestablish the mapping between hardware Ethernet addresses and IP addresses.

Gorry Fairhurst provides a nice description of ARP (see Resources).

The arp utility

The Linux utility arp lets you examine and modify the status of ARP mappings. A simple status report might look like Listing 1:

Listing 1. ARP status report

```
$ arp -n
Address          HWtype  HWaddress           Flags Mask    Iface
192.168.2.1      ether   00:03:2F:09:61:C7   C             eth0
```

This tells you the specific hardware device assigned to IP address 192.168.2.1 on

this network (the number used is suggestive of a router/gateway, which it is, in this case). The fact that only this single mapping is listed does not necessarily mean that no other devices exist on the local network, but simply that the ARP records for other devices have expired. ARP expires records after a short time -- on the order of minutes rather than seconds or hours -- to allow networks to reconfigure themselves if hosts are added or removed or if settings are changed on machines. By caching an ARP record for a short time, a new request should not be necessary during most client/server application sessions.

Any sort of IP request on a host that may be on the local network causes the kernel to send out an ARP request; if an ARP reply is received, add the host to the ARP cache (as in Listing 2):

Listing 2. Interacting with additional IP addresses

```
$ ping -c 1 192.168.2.101 > /dev/null
$ ping -c 1 192.168.2.101 > /dev/null
$ ping -c 1 192.168.2.102 > /dev/null
$ ping -c 1 192.168.32.32 > /dev/null
$ ping -c 1 192.168.32.32 > /dev/null
$ arp -n
Address          HWtype  HWaddress           Flags Mask    Iface
192.168.2.1      ether   00:03:2F:09:61:C7   C             eth0
192.168.2.101    ether   00:30:65:2C:01:11   C             eth0
192.168.2.100    ether   00:11:24:9D:1E:4B   C             eth0
192.168.2.102    ether   00:48:54:83:82:AD   C             eth0
```

In this case, the first four addresses really exist on the local Ethernet network, but 192.168.32.32 does not, so no ARP reply is received. Notice also that addresses you may succeed in connecting to via non-local routing will also not cause anything to be added to the ARP cache (see Listing 3):

Listing 3. Nothing added to the ARP cache

```
$ ping -c 1 google.com
PING google.com (216.239.57.99) 56(84) bytes of data.
64 bytes from 216.239.57.99: icmp_seq=1 ttl=235 time=109 ms
--- google.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 109.123/109.123/109.123/0.000 ms
$ arp -n
Address          HWtype  HWaddress           Flags Mask    Iface
192.168.2.1      ether   00:03:2F:09:61:C7   C             eth0
```

Google is *reachable* (because routing is already configured), but 216.239.57.99 is non-local and is not added to ARP. The seventh tutorial in this series, on topic 214, deals with network troubleshooting and demonstrates manually setting ARP.

PPP, PAP, and CHAP

Point-to-Point Protocol (PPP) is used to establish Internet links over dial-up modems, direct serial connections, DSL, and other types of point-to-point links (sometimes including PPPoE as a "pseudo-layer over Ethernet -- more of a handshake protocol). The pppd daemon works together with the kernel PPP driver to establish and maintain a PPP link with another system (called the peer) and to negotiate Internet Protocol (IP) addresses for each end of the link.

PPP, specifically pppd, authenticates its peer and/or supplies authentication information to the peer. Such authentication is performed using either the simple password system Password Authentication Protocol (PAP) or the per-session Challenge Handshake Authentication Protocol (CHAP). Of the two, CHAP is more secure if both ends support it.

Options for PPP in general are stored in /etc/ppp/options. Configuration of PAP is via the PAP secrets file /etc/ppp/pap-secrets; for CHAP it is in the CHAP secrets file /etc/ppp/chap-secrets.

The PAP/CHAP secrets file

The /etc/ppp/pap-secrets file contains whitespace-separated fields for *client*, *server*, *secret*, and *acceptable local IP address*. The last may be blank (and generally is for dynamic IP allocation). The PAP secrets file should be configured correspondingly for each peer. Even though PPP is a peer protocol, for connection purposes we call the requesting machine the client and the waiting machine the server. For example, the machine bacchus on my network might have a configuration like:

Listing 4. Configuring pap-secrets on the bacchus

```
# Every regular user can use PPP and uses passwords from /etc/passwd
# INBOUND connections
# client   server  secret            acceptable local IP addresses
*          bacchus ""                *
chaos      bacchus chaos-password
# OUTBOUND connections
bacchus      *     bacchus-password
```

The machine bacchus will accept connections claiming to be any regular user or also claiming to be the machine chaos (and demanding the password `chaos-password` in the latter case). To other machines, bacchus will simply use its own name and offer the password `bacchus-password` to every peer.

Correspondingly, the machine chaos on my network might have:

Listing 5. Machine chaos makes more conservative connection choices

```
# client   server  secret            acceptable local IP addresses
chaos      bacchus chaos-password
bacchus    chaos   bacchus-password
```

Machine chaos is more conservative regarding to whom it will connect. It is willing to exchange credentials only with bacchus. You may configure each /etc/ppp/options file to decide if credentials are demanded, though.

Using CHAP secrets requires that you allow for both peer machines to authenticate each other. As long as bi-direction authentication is configured in PAP secrets, a CHAP secrets file may look just the same as the above examples.

Connecting with mgetty

The PAP secrets file can be used with the AUTO_PPP function of mgetty. mgetty 0.99+ is preconfigured to start up pppd with the `login` option. This tells pppd to consult /etc/passwd (and /etc/shadow in turn) after a user has passed this file.

In general, a getty program may be configured to allow connections from serial devices, including modems and direct serial ports. For example, for a hard-wired line or a console tty, you might run:

```
/sbin/getty 9600 ttyS1
```

in your inittab. For a old style dial-in line with a 9600/2400/1200 baud modem:

```
/sbin/getty -mt60 ttyS1 9600,2400,1200.
```

Configuring routing

In the discussion of Address Resolution Protocol, we saw how IP addresses are assigned within a local network. However, to communicate with machines outside of a local network, it is necessary to have a *gateway/router*. Basically, a gateway is simply a machine that connects to more than one network and can therefore take packets transmitted within one network and re-transmit them on other networks it is connected to. This is where the name "Internet" comes from: It is a "network of networks" in which every gateway can eventually reach every other network that is said to be "on the Internet."

The fifth tutorial in this series, on topic 210, deals with network client management, and discusses DHCP. DHCP will assign both client IP addresses and gateway addresses. However, with a fixed IP address on a client or in debugging situations, the Linux command `route` allows you to view and modify routing tables. The newer command `ip` also lets you modify routing tables using a somewhat more powerful syntax.

A routing table simply lets you determine which gateway or host to send a packet to, given a specific pattern in the address. An address pattern is specified by combining an address with a *subnet mask*. A subnet mask is a bit pattern, usually represented in dotted quad notation, that tells the kernel which bits of a destination to treat as the *network address* and which remaining bits to treat as a *subnet*. The command `ip` can accept the simpler */NN* format for bitmasks. In general, in a mask and address, zero bits are "wildcards."

For example, a simple network with a single external gateway is likely to have a routing table similar to Listing 6:

Listing 6. Typical simple routing table

```
$ route -n
Kernel IP routing table
Destination  Gateway      Genmask        Flags Metric Ref Use Iface
192.168.2.0  0.0.0.0      255.255.255.0  U     0      0     0 eth0
0.0.0.0      192.168.2.1  0.0.0.0        UG    0      0     0 eth0
```

What this means is simply that any IP address that matches "192.168.2.*" is on the local network and will be delivered directly to the proper host (resolved with ARP). Any other address will be sent to the gateway "192.168.2.1", which will be required to forward a packet appropriately. The machine 192.168.2.1 must be connected to one or more external networks.

However, for a more complex case, you may route specific patterns differently. In an invented example, let's say that you want to route specific */16* addresses through other gateways. You could do what is shown in Listing 7:

Listing 7. Changing routing on /16 networks

```
$ route add -net 216.109.0.0 netmask 255.255.0.0 gw 192.168.2.2
$ route add -net 216.239.0.0 netmask 255.255.0.0 gw 192.168.2.3
$ route -n
Kernel IP routing table
Destination  Gateway      Genmask        Flags Metric Ref Use Iface
192.168.2.0  0.0.0.0      255.255.255.0  U     0      0     0 eth0
216.109.0.0  192.168.2.2  255.255.0.0    UG    0      0     0 eth0
216.239.0.0  192.168.2.3  255.255.0.0    UG    0      0     0 eth0
0.0.0.0      192.168.2.1  0.0.0.0        UG    0      0     0 eth0
```

Addresses of the form "216.109.*" and "216.239.*" will now be routed through the gateways 192.168.2.2 and 192.168.2.3, respectively (both on the local network themselves). Addresses that are local, or outside the pattern spaces given, will be routed as before. You can use the command `route delete` correspondingly to remove routes.

Section 4. Advanced network configuration and troubleshooting

About network utilities

Linux comes with a number of standard utilities you can use to customize and troubleshoot a network configuration. Although much of Linux networking code lives in the kernel itself, almost everything about the behavior of networks is configurable using command-line utilities. Many distributions also come with higher level and/or graphical high-level configuration tools, but those do not contain anything that can not be performed and scripted using the command-line tools.

The ping utility

The most basic way to check whether a Linux host has access to an IP address (or to a named host, once DNS and/or /etc/hosts is configured) is with the utility **ping**. ping operates at the basic IP layer and it does not rely on the Data Link Layer like TCP or IP. ping instead uses the Internet Control Message Protocol (ICMP). If you

cannot reach a host with ping, you can assume you will not reach it with any other tool, so ping is always the first step in establishing whether a connection to a host is available (man ping has details on options).

By default, ping sends a message every one second until cancelled, but you may change timings, limit message count, and output details. When ping is run, it returns some details on round-trip time and dropped packages, but for the most part either you can ping a host or you cannot. Listing 8 gives some examples:

Listing 8. Local and non-local ping examples

```
$ ping -c 2 -i 2 google.com
PING google.com (216.239.37.99): 56 data bytes
64 bytes from 216.239.37.99: icmp_seq=0 ttl=237 time=43.861 ms
64 bytes from 216.239.37.99: icmp_seq=1 ttl=237 time=36.956 ms

--- google.com ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 36.956/40.408/43.861 ms

$ ping 192.168.2.102
PING 192.168.2.102 (192.168.2.102): 56 data bytes
64 bytes from 192.168.2.102: icmp_seq=0 ttl=255 time=4.64 ms
64 bytes from 192.168.2.102: icmp_seq=1 ttl=255 time=2.176 ms
^C
--- 192.168.2.102 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 2.176/3.408/4.64 ms
```

The iconfig utility

Network interfaces are configured with the tool **ifconfig**. Usually, this is run as part of the initialization process, but in some cases interfaces may be modified and tuned later (especially for debugging). If you run ifconfig with no switches, it displays the current status. You may use the forms `ifconfig <interface> up` and `ifconfig <interface> down` to start and stop network interfaces. Some other switches change the display or limit the display to specific interfaces. See man ifconfig for more details.

An informational display might look something like Listing 9:

Listing 9. Using ifconfig to examine network interfaces

```
$ ifconfig
eth0  Link encap:Ethernet  HWaddr 00:12:F0:21:4C:F8
      inet addr:192.168.2.103  Bcast:192.168.2.255  Mask:255.255.255.0
      inet6 addr: fe80::212:f0ff:fe21:4cf8/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:540 errors:0 dropped:0 overruns:0 frame:0
      TX packets:233 errors:0 dropped:0 overruns:0 carrier:1
      collisions:0 txqueuelen:1000
      RX bytes:49600 (48.4 KiB)  TX bytes:42067 (41.0 KiB)
      Interrupt:21 Base address:0xc000 Memory:ffcfe000-ffcfefff

ppp0  Link encap:Point-Point Protocol
      inet addr:10.144.153.104  P-t-P:10.144.153.51 Mask:255.255.255.0
      UP POINTOPOINT RUNNING  MTU:552  Metric:1
      RX packets:0 errors:0 dropped:0 overruns:0
      TX packets:0 errors:0 dropped:0 overruns:0


lo    Link encap:Local Loopback
```

```
       inet addr:127.0.0.1  Mask:255.0.0.0
       inet6 addr: ::1/128 Scope:Host
       UP LOOPBACK RUNNING  MTU:16436  Metric:1
       RX packets:4043 errors:0 dropped:0 overruns:0 frame:0
       TX packets:4043 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:0
       RX bytes:368044 (359.4 KiB)  TX bytes:368044 (359.4 KiB)
```

In this display, two networks are configured, one on Ethernet and one on PPP (plus the local loopback). In other cases, you might have multiple Ethernet interfaces configured or other interface types. If so, the system is called *multi-homed*.

The netstat utility

Linux utilities tend to overlap in functionality. The tool **netstat** displays information that may also be provided by several utilities, such as ifconfig and route. You may also find extensive general statistics on network activity. For example:

Listing 10. Network statistics report

```
$ netstat -s
Ip:
    12317 total packets received
    0 forwarded
    0 incoming packets discarded
    12255 incoming packets delivered
    11978 requests sent out
Icmp:
    1 ICMP messages received
    0 input ICMP message failed.
    ICMP input histogram:
       echo replies: 1
    0 ICMP messages sent
    0 ICMP messages failed
    ICMP output histogram:
Tcp:
    7 active connections openings
    5 passive connection openings
    0 failed connection attempts
    0 connection resets received
    3 connections established
    11987 segments received
    11885 segments send out
    0 segments retransmitted
    0 bad segments received.
    3 resets sent
Udp:
    101 packets received
    0 packets to unknown port received.
    0 packet receive errors
    92 packets sent
TcpExt:
    1 TCP sockets finished time wait in fast timer
    1490 delayed acks sent
    Quick ack mode was activated 5 times
    3632 packets directly queued to recvmsg prequeue.
    126114 of bytes directly received from backlog
    161977 of bytes directly received from prequeue
    1751 packet headers predicted
    3469 packets header predicted and directly queued to user
    17 acknowledgments not containing data received
    4696 predicted acknowledgments
    0 TCP data loss events
```

Other utilities

There are several other utilities you should be aware of for network configuration. As usual, their respective manpages contain full usage information. Detailed discussion of these is in the seventh tutorial in this series, on topic 214, which deals with network troubleshooting.

tcpdump lets you monitor all the packets that pass through network interfaces, optionally limited to particular interfaces or filtered on various criteria. Often saving this packet summary information, then filtering or summarizing it with text-processing tools, is useful for debugging network problem. For example, you can examine packets communicated with a particular remote host.

lsof lists open files on a running Linux system. But in particular, you may use the `lsof -i` option to examine only the pseudo files for a particular IP connection or for network connections in general. For example:

Listing 11. Using lsof to examine pseudo files for connections

```
$ lsof -i
        COMMAND    PID USER   FD   TYPE DEVICE SIZE NODE
                NAME
        vino-serv 7812  dqm   33u  IPv4  12824
                TCP *:5900 (LISTEN)
        gnome-cup 7832  dqm   18u  IPv4  12865
                TCP localhost.localdomain:32771->localhost.localdomain:ipp (ESTABLISHED)
        telnet    8909  dqm    3u  IPv4  15771
                TCP 192.168.2.103:32777->192.168.2.102:telnet (ESTABLISHED)
```

nc and **netcat** are aliases. netcat is a simple UNIX utility that reads and writes data across network connections using TCP or UDP protocol. It is a "back-end" tool that can be used directly or driven by other programs and scripts. In many respects, netcat is similar to telnet, but is more versatile in allowing UDP interaction and sending unfiltered binary data.

Resources

Learn

- Review the entire LPI exam prep tutorial series on developerWorks to learn Linux fundamentals and prepare for system administrator certification.
- At the LPIC Program, find task lists, sample questions, and detailed objectives for the three levels of the Linux Professional Institute's Linux system administration certification.
- Read Kwan Lowe's Kernel Rebuild Guide for more details on building a kernel.
- Learn how applications use TCP or UDP in these tutorials:
  - "Programming Linux sockets, Part 1" (developerWorks, October 2003)
  - "Programming Linux sockets, Part 2" (developerWorks, January 2004)
- *TCP/IP Network Administration, Third Edition* by Craig Hunt (O'Reilly, April 2002) is an excellent resource on Linux networking.
- Find Ether number assignments at IANA.
- Gorry Fairhurst provides a nice description of ARP.
- Many Linux User Groups have local and distance study groups for LPI exams -- here's a list or more than 700 LUGs around the world.
- The Linux Documentation Project has a variety of useful documents, especially its HOWTOs.
- Find more tutorials for Linux developers in the developerWorks Linux zone.

Get products and technologies

- Get the Linux kernel source at the Linux Kernel Archives.
- Order the SEK for Linux, a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- Build your next development project on Linux with IBM trial software, available for download directly from developerWorks.

Discuss

- Participate in the discussion forum for this content.
- Get involved in the developerWorks community by participating in developerWorks blogs.

About the author

David Mertz, Ph.D.

David Mertz has been writing the developerWorks columns *Charming Python* and *XML Matters* since 2000. Check out his book *Text Processing in Python* . For more on David, see his personal Web page.