LPI exam 201 prep: Troubleshooting

Intermediate Level Administration (LPIC-2) topic 214

Skill Level: Intermediate

Brad Huntting (huntting@glarp.com)
Mathematician
University of Colorado

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

02 Sep 2005

In this tutorial, Brad Huntting and David Mertz continue preparing you to take the Linux Professional Institute® Intermediate Level Administration (LPIC-2) Exam 201. The last of eight tutorials, this tutorial focuses on what you can do when things go wrong. It builds on material already covered in more detail in earlier 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 junior and intermediate levels. To attain each level of certification, you must pass two LPI exams.

Each exam covers several topics, and each topic has a weight. The weights indicate the relative importance of each topic. Very roughly, expect more questions on the exam for topics with higher weight. The topics and their weights for LPI exam 201 are:

Topic 201

Linux kernel (weight 5).

Topic 202
System startup (weight 5).

Topic 203
File systems (weight 10).

Topic 204
Hardware (weight 8).

Topic 209
File and service sharing (weight 8).

Topic 211
System maintenance (weight 4).

Topic 213
System customization and automation (weight 3).

Topic 214
Troubleshooting (weight 6). The focus of this tutorial.

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 "Troubleshooting," the last of eight tutorials designed to prepare you for LPI exam 201. In this tutorial, you review material already covered in more detail in earlier tutorials but with a focus on learning what to do when things go wrong.

The tutorial is organized according to the LPI objectives for this topic, as follows:

2.214.2 Creating recovery disks (weight 1)
You will be able to create both a standard bootdisk for system entrance and a recovery disk for system repair.

2.214.3 Identifying boot stages (weight 1)
You will be able to determine, from bootup text, the four stages of boot sequence and distinguish between each.

2.214.4 Troubleshooting LILO (weight 1)
You will be able to determine specific stage failures and corrective techniques.

2.214.5 General troubleshooting (weight 1)
You will be able to recognize and identify boot loader- and kernel-specific stages and utilize kernel boot messages to diagnose kernel errors. This objective includes identifying and correcting common hardware issues, and determining if the problem is hardware or software.

2.214.6 Troubleshooting system resources (weight 1)
You will be able to identify, diagnose, and repair your local system environment.

2.214.8 Troubleshooting environment configurations (weight 1)
You will be able to identify common local system and user environment configuration issues and common repair techniques.

Troubleshooting a Linux system that is not working -- or more especially, is not *entirely* working -- is probably the most frustrating part of system administration. Linux is not special in this regard. With Linux, common tasks -- adding new hardware or software, tweaking the behavior of installed software, or scripting and scheduling routine operations -- are straightforward. But when a running system suddenly stops, you are presented with a dizzying collection of things that *might* be wrong.

This tutorial cannot solve all real-world problems, but it does point to tools that are helpful in the troubleshooting process (most touched briefly on in other tutorials) and reminds you of the best places to look for sources of trouble. In troubleshooting, a good understanding of Linux, combined with patience and elbow grease, will go a long way.

Prerequisites

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

Section 2. Creating recovery disks

Before you recover

When, in course of system events, a Linux installation gets so messed up that it just cannot boot normally, a good first course of action is to attempt to "boot single-user" to fix the problem. Single-user mode (runlevel 1) allows you to access and modify many problems without worrying about permission issues or files being locked by other users and processes. Moreover, single-user mode runs with a minimum of services, daemons, and tasks that might confound -- or even be the cause of -- your underlying problem.

In some cases, however, even files needed for single-user mode such as /etc/passwd, /etc/fstab, /etc/inittab, /sbin/init, /dev/console, etc., can become corrupted and the system will not boot, even into single-user mode. In times like this, a recovery disk (a bootable disk which contains the essentials needed to repair a broken partition, sometimes called a *repair disk*) can be used to get the system back on its feet.

Recovery CDs

Today, standalone CD (or DVD) distributions such as Knoppix can be used as the "Cadillac" of recovery disks, sporting a vast array of drivers, debugging tools, and even a Web browser. Even if you have special needs such as unusual drivers (like kernel modules) or recovery software, you are usually best off downloading a Linux LiveCD like Knoppix and customizing the contents of the ISO image before burning a recovery disk. See the tutorial for Topic 203 for information on loop mounting an ISO image.

Recovery floppies

For very old systems without bootable CD-ROMs, floppies containing only the bare essentials have to be used for recovery. Niceties like *emacs* and *vim* are typically too bulky to fit on a single recovery floppy, so an experienced system administrator should be prepared to use a stripped-down editor like *ed* to fix corrupted configuration files.

To create custom boot floppies, the user should read the Linux Documentation Project's "Linux-Bootdisk HOWTO." In many cases, however, Linux distributions will offer to create a recovery diskettes at installation time or later using included tools. Be sure to create and store recover disks *before* you really need them!

Section 3. Identifying boot stages

About booting

The tutorial on Topic 202 contains much more extensive information about Linux boot sequences. We will review the stages in quick summary for this tutorial.

The first stage of bootup has changed little since IBM®-compatible PCs with hard disks were first introduced. The BIOS reads the first sector of the boot disk into memory and runs it. This 512-byte Master Boot Record (MBR), which also stores the fdisk label, turns around and loads the "OS loader" (either GRUB or LILO) from the "active" partition.

Booting the kernel

Once the system BIOS on an x86 system (other architectures vary slightly) runs the MBR, several steps happen for the Linux kernel to load. If the boot does not succeed, determining where it fails is the first step in figuring out what needs to be fixed.

- Load Boot Loader (LILO/Grub).
- Boot loader starts and hands off control to kernel.
- Kernel: Load base kernel and other essential kernel modules.
- Hardware initialization and setup:
  - Kernel starts.
  - Initialize kernel infrastructure (VM, scheduler, etc.).
  - Device Drivers Probe and Attach.
  - Root filesystem mounted.

Booting the userland

Assuming a base kernel, kernel modules, and root filesystem start successfully (or at least successfully enough not to freeze completely), the system initialization process starts:

- Daemon initialization and setup
  - /sbin/init started as process 1
  - /sbin/init reads /etc/inittab
  - /etc/rc<n>.d/S* scripts are run by /etc/init.d/rc
  - Disks fsck'ed
  - Network interfaces configured
  - Daemons started
  - getty(s) started on console and serial ports

Section 4. Troubleshooting LILO and GRUB

Looking for the boot loader

When GRUB starts up it flashes "GRUB loading, please wait..." on the screen and then jumps into its boot menu. LILO usually displays a `LILO:` prompt (but some versions jump directly to a menu). If you do not reach a LILO or GRUB screen, it is probably time for a separate recovery disk. Generally, you will know you have this problem with some kind of BIOS message about not finding a boot sector or maybe even not finding a hard disk.

If your system cannot locate a usable boot sector, it is possible that your LILO or GRUB installation became corrupted. Sometimes other operating systems step on boot sectors (or overzealous "virus protection programs" on Windows®) and occasionally other problems arise. A recovery disk will let you reinstall LILO or GRUB. Typically you will want to `chroot` to your old root filesystem to utilize your prior LILO or GRUB configuration files.

In the hard-disk-not-found case, hardware failure is likely (and you will hope you made a backup); often in the no-boot-sector case too. See the tutorial on Topic 202 for more information on LILO and GRUB.

Configuring LILO and GRUB

LILO is configured with the file /etc/lilo.conf or by another configuration file specified at the command line. To reinstall LILO, first make sure your lilo.conf file really matches your current system configuration (the right partition and disk number information especially; this can become mixed up if you swap hard disk and/or change partitions). Once you have verified your configuration, simply run /sbin/lilo as root (or in single-user mode).

Configuring and reinstalling GRUB is essentially the same process as for LILO. GRUB's configuration file is /boot/grub/menu.lst, but it is read each time the system boots. Likewise, the GRUB shell lives at /boot/grub/, but you can choose which hard disk the basic GRUB MBR lives at with the `groot=` option in /boot/grub/menu.lst. To install GRUB to an MBR, run a command like `grub-install /dev/hda`, which will check the currently mounted /boot/ for its configuration.

Section 5. General troubleshooting

The Linux filesystem structure

Linux distributions vary a bit in where they put files. The Linux Filesystem Hierarchy Standard is making progress in furthering standardization of this. A few directories are already well standardized and are particularly important to look at for startup and runtime problems:

- /proc/ is the virtual filesystem with information on processes and system status. All the guts of a running system live here, in a sense. See Topic 201 for more information.

- /var/log/ is where the log files live. If something goes wrong, it is good that helpful information is in some log file here.
- / is generally the root of the filesystem under Linux which simply contains other nested directories. On some systems bootup files like vmlinuz and initrd.img might be here rather than in /boot/.
- /boot/ stores files directly used in the kernel boot process.
- /lib/modules/ is where kernel modules live, nested under this directory in subdirectories named for the current kernel version (if you boot multiple kernel versions, multiple directories should exist). For example:
  ```
  % ls /lib/modules/2.6.10-5-386/kernel/
  arch crypto drivers fs lib net security sound
  ```

Finding bootup messages

During actual system boot, messages can scroll by quickly and you may not have time to identify problems or unexpected initialization activities. Some information of interest is likely to be logged with syslog, but the basic kernel and kernel module messages can be examined with the tool *dmesg*.

Hardware/system identification tools

The tutorial for Topic 203 (Hardware) contains more information on hardware identification. Generally, you should keep in mind these system tools:

- lspci: Lists all PCI devices.
- lsmod: Lists loaded kernel modules.
- lsusb: Lists USB devices.
- lspnp: Lists Plug-and-Play devices.
- lshw: Lists hardware.

Slightly farther from hardware, but still useful:

- lsof: Lists open files.
- insmod: Loads kernel modules.
- rmmod: Removes kernel modules.
- modprobe: Higher level insmod/rmmod/lsmod wrapper.
- uname: Prints system information (kernel version, etc.).

- strace: Traces system calls.

When you get desperate in trying to use binaries, whether libraries or applications, the tool *strings* can really rescue you (but expect to work at it). Crucial information like hard-coded paths are sometimes buried in binaries and with a good measure of trial-and-error, you can find them by looking for the strings within binaries.

Section 6. Troubleshooting system resources and environment configurations

The initialization

As Topic 202 addresses in more detail, the startup of Linux after the kernel loads is controlled by the init process. The main controlling configuration for init is /etc/inittab. The /etc/inittab contains details on what steps to perform at each runlevel. But perhaps the most crucial thing it does is actually set the runlevel for later actions. If your system is having troubles booting, a setting a different runlevel may help the process. The key line is something like:

```
# The default runlevel. (in /etc/inittab)
id:2:initdefault:
```

Resource scripts

The rc scripts run at boot time, shutdown time, and whenever the system changes runlevels and they are responsible for starting and stoping most system daemons. In most (read *modern*) Linux distributions, the rc scripts live in the directory /etc/init.d/ and are linked to the directories /etc/rc<N>.d/ (for *N=runlevel*) where they are named "S*" for startup scripts and "K*" for shutdown scripts. The system never runs the scripts from /etc/init.d directly, but looks for them in /etc/rc<N>.d/[SK]*.

The system shell

Rarely, but conceivably, a system administrator may wish to change the system-wide shell startup script /etc/profile. Such a change affects every interactive shell (except for /bin/tcsh users or other non-/bin/sh-compatible shells). Corrupting this file can easily lead to a situation where no one can login and a recovery disk may be needed to rectify the situation. The normal method of changing shell behavior on an individual basis is by modifying /home/$USER/.bash_profile and /home/$USER/.bashrc.

Configure kernel parameters

The *sysctl* system (see the manpage for sysctl) was taken from BSD UNIX® and is be used to tune some system resources. Run `sysctl -a` to see what variables can be controlled by sysctl and what they are set to. The sysctl utility is most useful to tuning networking parameters as well as some kernel parameters. Use the file /etc/sysctl.conf set sysctl parameters at boot time.

Dynamic libraries

On most systems, dynamic libraries are constantly being added, updated, replaced, and removed. Since almost every program on the system needs to find and load dynamic libraries, the names, version numbers, and locations of most dynamic libraries are cached by the *ldconfig* program. Normally only the dynamic libraries in the system directories /lib/ and /usr/lib/ are cached. To add more directories to the system-wide default search path, add the directory names (such as /usr/X11R6/lib) to the /etc/ld.so.conf and run ldconfig as root.

System logging

Topic 211 discusses syslog in detail. The key file to keep in mind if you have problems (and especially if you want to make sure you can analyze them later) is /etc/syslog.conf. By changing the contents of this configuration, you have granular control over what events are logged and where they are logged -- including potentially using mail or remote machines. If problems occur, make sure the subsystems where you think they arise log information in a manner you can readily examine forensically.

Periodic events

On almost every Linux system, the cron daemon is running. See Topic 213 for more details on cron and crontab. In general, a source of potential problems to keep in mind is scripts that are run intermittently. It is possible that what you believe to be a kernel or application issue results from unrelated scripts that are run behind the scenes by cron.

An extreme measure is to kill cron altogether. You can find its process nubmer with `ps ax | grep cron` and `kill` can terminate it. A less drastic measure is to edit /etc/crontab to run a more conservative collection of tasks; also be sure to constrain the user tasks that are scheduled by editing /etc/cron.allow and /etc/cron.deny. Even though users usually should not have enough permissions to cause system-wide problems, a good first step is to temporarily disable user crontab configurations and see if that resolves problems.

Resources

Learn

- 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.
- "Linux hardware stability guide" (developerWorks, March 2001) is a two-part guide on troubleshooting Linux hardware.
- "Common threads: Advanced filesystem implementor's guide, Parts 1 - 13" (developerWorks, starting June 2001) is an excellent series on Linux filesystems.
- "Understanding Linux configuration files" (developerWorks, December 2001) explains configuration files on a Linux system that control user permissions, system applications, daemons, services, and other administrative tasks in a multi-user, multi-tasking environment.
- "System recovery with Knoppix" (developerWorks, October 2003) shows how to access a non-booting Linux system with a Knoppix CD.
- "Assess system security using a Linux LiveCD" (developerWorks, July 2005) introduces four LiveCD offerings that specialize in nailing down vulnerabilities.
- Read the Linux Documentation Project's Linux-Bootdisk HOWTO.
- Find more resources for Linux developers in the developerWorks Linux zone.

Get products and technologies

- Read about and download Knoppix.
- 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 authors

Brad Huntting
Brad has been doing UNIX systems administration and network engineering for about 14 years at several companies. He is currently working on a Ph.D. in Applied Mathematics at the University of Colorado in Boulder, and pays the bills by doing

UNIX support for the Computer Science department.

David Mertz, Ph.D.
David Mertz is Turing complete, but probably would not pass the Turing Test. For more on his life, see his personal Web page. He's been writing the developerWorks columns *Charming Python* and *XML Matters* since 2000. Check out his book *Text Processing in Python* .