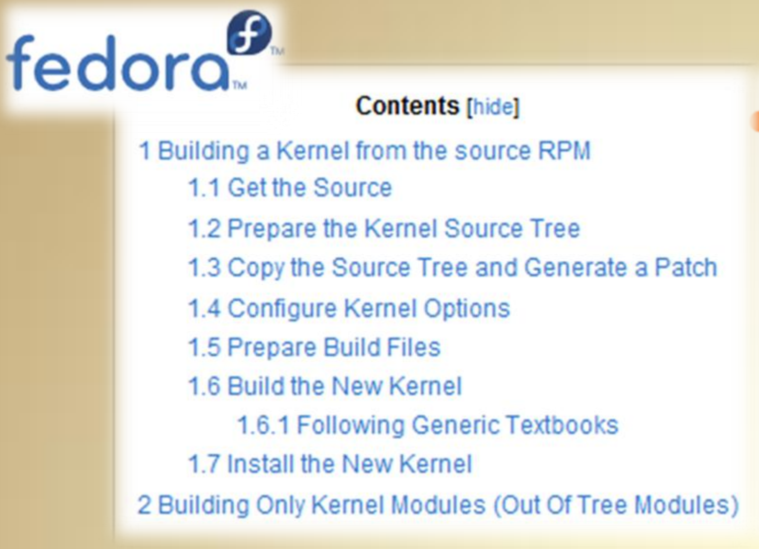
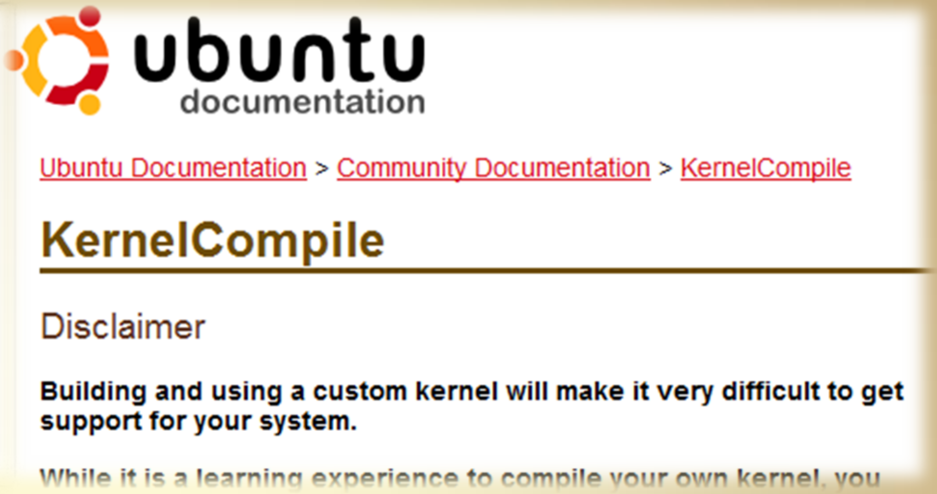


Contents [hide]



Ubuntu Documentation > Community Documentation > KernelCompile

KernelCompile

Disclaimer

Building and using a custom kernel will make it very difficult to get support for your system.

While it is a learning experience to compile your own kernel, you

Kernel Configuration

This material is based on work supported by the National Science Foundation under Grant No. 0802551

Lesson Overview

Linux operating systems have a core component called the kernel which functions as the heart of the system and determines how the computer interacts with hardware and software. As a Linux administrator, you may be asked to update, fix, or re-build the Linux kernel. These changes to the kernel are often necessary to improve efficiency, add functionality, or remove features to enhance security and stability.

In this lesson, you will explore the Linux kernel and the various methods to compile, package , and install it. By the end of this lesson, you should know how to setup a build system, install and configure the kernel , and build RPM and Debian-based kernel packages.

This topic is important because every Linux administrator must have a thorough understanding of the Linux kernel in order to make necessary changes or improvements to the core system. At the end of this lesson, you will have a better grasp of kernel configuration and the best practices for kernel management.

Objectives – Student Expectations

Student Expectations

You should know what will be expected of you when you complete this lesson. These expectations are presented as objectives.

Objectives are short statements of expectations that tell you what you must be able to do, perform, learn, or adjust after reviewing the lesson.

Objectives

Objective

Given the need to add new functionality to the Linux kernel, or revise the configuration of the Linux kernel, the student will be able to compile and boot a Linux kernel successfully as per industry standards.

Lesson Outline

During this lesson, you will explore:

- The purpose of the Linux kernel
- The Linux kernel build system
- Kernel management
- Diff and Patch files
- Configuration of the Linux kernel
- Building and installing a kernel RPM Package
- Building and installing a kernel DEB Package
- Building the kernel manually for test purposes

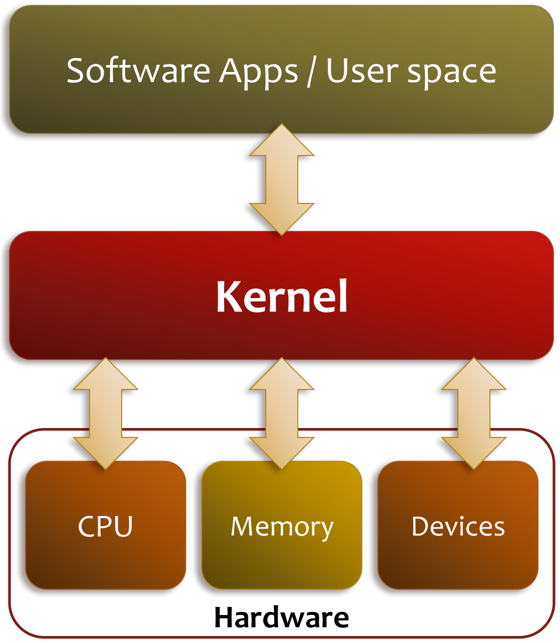
What is the Linux Kernel?

The Linux kernel is a piece of software that forms the core or center of the Linux operating system. It controls the communication between the hardware devices on a computer system and software applications. It also handles memory and CPU priorities. The kernel sits between the "user space" (the area where software applications are run) and the hardware components and ensures they work together.

If computer users wish to open a file stored on the hard drive, or communicate with a hardware device, they will use a software application to make the request. When the command to open the file is issued, the command is routed through the kernel. In this way, the software is able to access the attached hardware (disk drives, USB sticks, keyboards, and anything else connected to the computer.

The kernel serves as a mediator between your software and hardware. It makes sure that everyone plays fair, gets the memory and the processor time required, and performs necessary tasks at the right time. It is a portable interface between platforms.

The kernel contains code that "talks" to the hardware devices (i.e. memory, video cards, sound cards, and disk drives). The kernel handles and tracks all memory related functions.



Required Reading

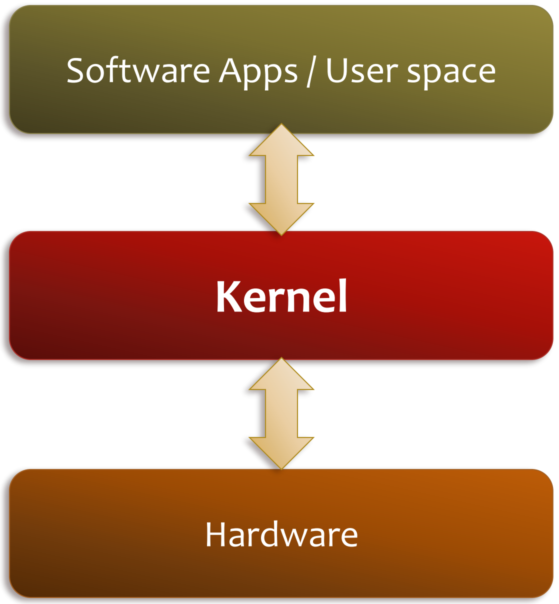
- Anatomy of Kernel
- Linux boot process
- Linux Kernel

Linux Kernel

The kernel is modular, which means that it can be compiled to include or not include hardware modules which the user needs or does not need. This also means that the kernel can be extremely efficient or extremely bloated. The size of the kernel depends on the skill of the Linux Administrator.

The Linux kernel is open source, which means that all the code is available to the end user. If you have a problem with a driver, have some coding knowledge, and want to fix it, you can. Just send the patches to Linus Torvalds and they may be included in the next kernel release!

The Linux kernel has seven sections. Each will be explored in turn.
- System call interface (SCI)
- Process management (PM)
- Virtual file system (VFS)
- Memory management (MM)
- Network stack
- Architecture dependent code
- Device Drivers (DD)



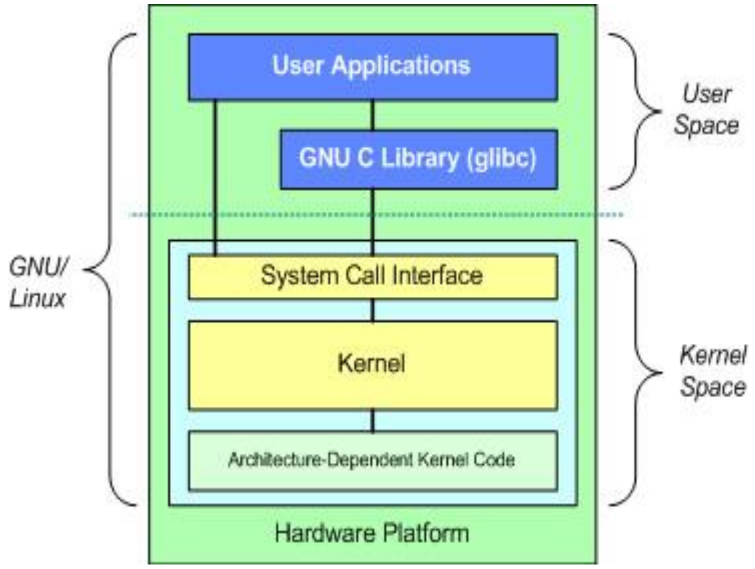
Suggested Reviewing
- The Linux Kernel
- Kernel Diagram

SCI and Process Management

One of the seven components of the Linux kernel is the System Call Interface (SCI). The SCI is a thin kernel layer that provides the interface between the user space and the kernel. In Linux, this layer is architecture dependent, meaning that the type of the processor used in the system matters. Please review IBM's explanation of the SCI.

Process management is another component of the kernel. A process is an individual task that the processor is going to perform. This task is sometimes referred to as a thread. The process management layer of the Linux kernel is responsible for the execution of these processes.

The processes may or may not need CPU time. If CPU time is needed, the process management layer of the Linux kernel allows two or more different tasks to use the CPU. The process management layer of the kernel also is responsible for stopping a process, forking a process (creating a new process from an old one), and communicating between different processes or threads. Use the **top** command to view threads or processes running on a machine.



Required Reading

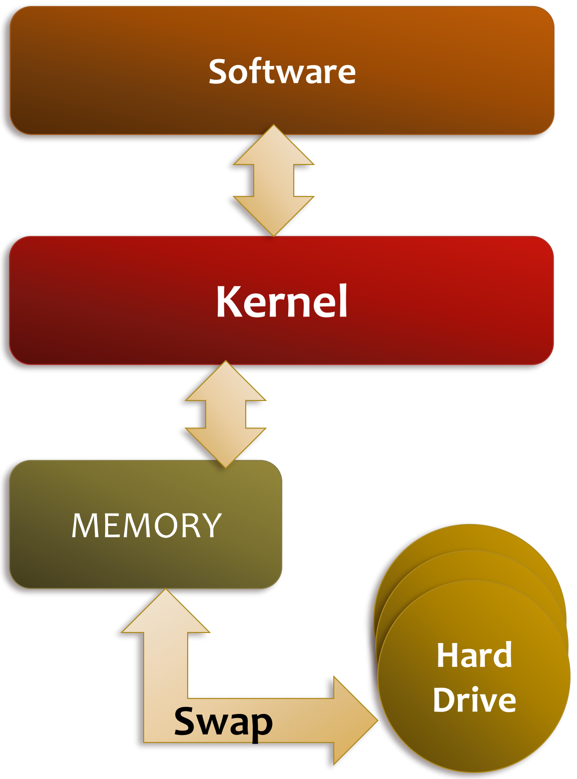
- The Linux Kernel
- Using the /Proc filesystem
- Inside the Linux scheduler

Memory Management

A third component of the Linux kernel is memory management.

The kernel also manages the memory use of the computer. The memory installed on the system, including virtual or disk-based memory, is managed in pages. These pages are 4KB allocated buffers of memory (or 4KB chunks of memory). The memory management layer of the kernel keeps track of which pages are full, partially full, or empty.

When multiple processes (or programs) need to use the same segments of memory, these pages can be moved out of the hardware memory and saved to the hard disk. This movement is called swapping—the pages of memory are swapped onto the hard drive. When the other processes need the memory again, the reverse process occurs and the saved pages are moved from the hard drive back into the physical memory. Consequently, having more physical memory on a computer system may decrease the frequency of page swaps.



Required Reading

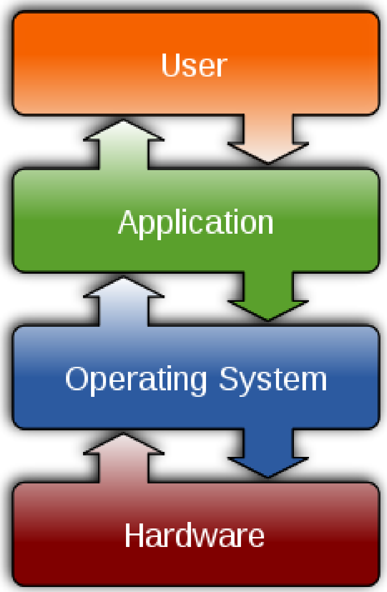
- Dynamic memory management

Virtual Filesystem & Network Stack

Virtual File system

In addition to system calls, process and memory management, the Linux kernel also creates a virtual file system that serves as the common interface for all file systems accessed by client applications. This virtual file system provides a common interface between the kernel's SCI and the supported file system.

This VFS interface allows the kernel to rapidly communicate with multiple file system types which are considered "plug-ins" to the kernel functionality.



Network Stack

The network stack provides the interface between the Internet Protocol (IP) and the system call interface (SCI)of the kernel.

Required Reading

- Virtual File System
- Virtual file systems
- Better networking with SCTP
- Linux networking stack
- The journey of a packet

Device Drivers & Architecture Code

Device Drivers

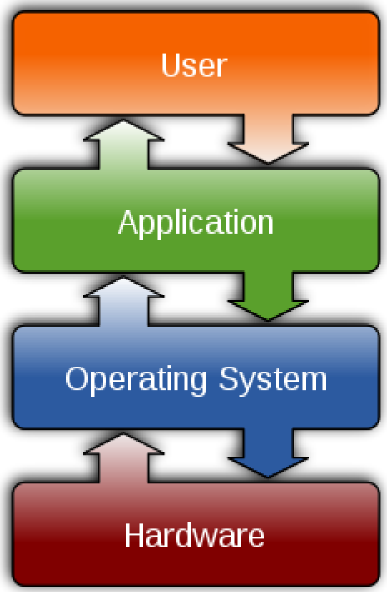
The device driver layer is the portion of the kernel containing the largest amount of source code. This is the layer that makes a particular hardware device useable to the computer system. For example, if you wish to plug-in a specific brand of USB keyboard, the device driver for that keyboard brand must be available in the kernel.

In the Linux kernel source code, the device drivers are divided into sub-directories based on the type of the driver.

Architecture-dependent code

One of the biggest selling points of Linux is that it will run on almost any type (or architecture) hardware. For example, Linux will run on the Intel based x86 family or processors, the AMD64 processors, the PPC processors found on older Apple computers, and many more.

The architecture dependent code in the Linux kernel is a small section of code that provides the interface that makes this work. A kernel compiled for an Intel 386 machine will not work on a PPC machine because of the different architecture dependent code that is included.



Required Reading

- Drivers demystified
- Device drivers
- Writing device drivers
- General device drivers

What is Kernel Configuration?

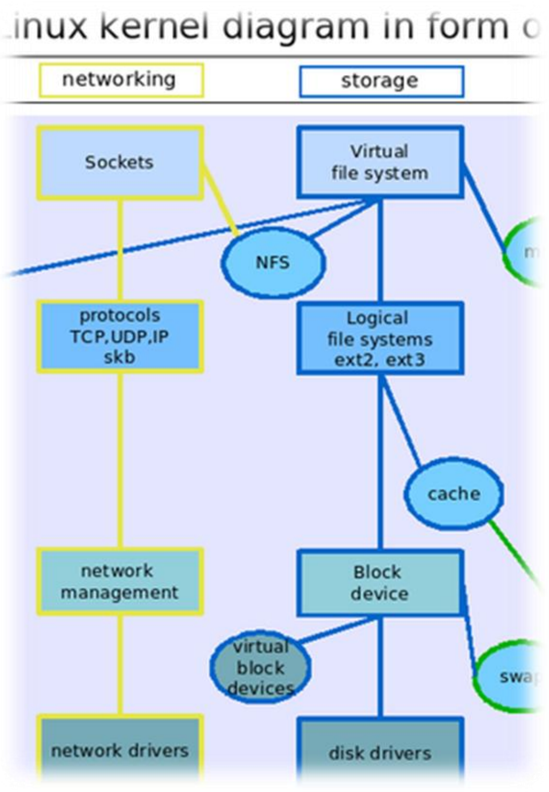
One of the first steps of kernel management is kernel configuration. Kernel configuration is the manual process of deciding what the kernel's functionality will be when it is compiled.

This process includes deciding the architecture on which it will run, how memory and processes are managed, what networking protocols will be available, what file system types may be used, and what device drivers will be available. Each of the seven layers of the Linux kernel is affected during kernel configuration.

As the Linux administrator, you may be called upon to configure the kernel to include new code, enhance security, enhance system speed, or correct mistakes. It is important to recognize what layer of the kernel you are being asked to change when such a request is made. You must also keep track of the version numbers of the kernel used in your environment.

Required Reading

- Configuring the kernel
- Kernel configuration file



Select image for a simplified view of the Linux kernel.

Kernel Version Numbers

The kernel version numbers have a purpose. They identify the components of the kernel and help to identify the most recent version. Version numbers are also important for bug tracking and development.

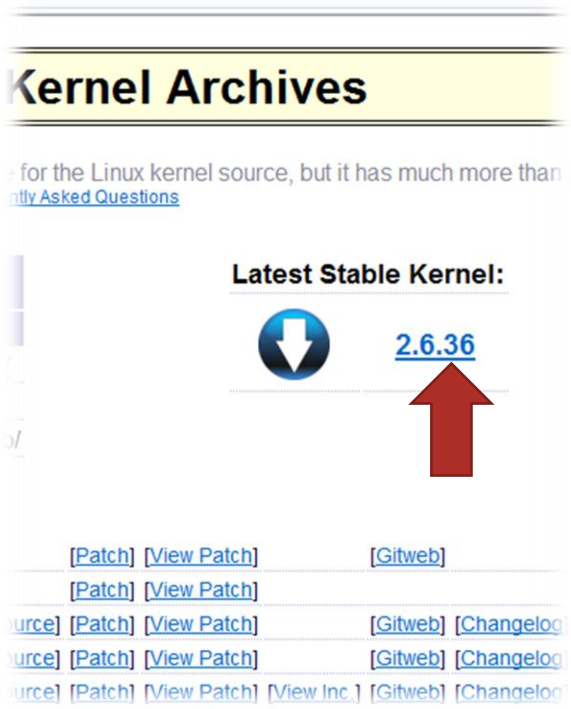
The production (stable) versions of the kernel have even numbers:

- 2.2
- 2.4
- 2.6

The development (unstable) versions of the kernel have odd numbers:

- 2.3
- 2.5
- 2.7

The third number in the kernel is the release number. When compiling a kernel always try to use the highest version and release of source code available as it contains the most recent fixes and enhancements. It is a good practice to subscribe to the Linux kernel mailing list to keep abreast of the latest changes.



ernel Archives

for the Linux kernel source, but it has much more than
tly Asked Questions

Latest Stable Kernel:
2.6.36

[Patch] [View Patch] [Gitweb]
[Patch] [View Patch]
urce] [Patch] [View Patch] [Gitweb] [Changelog
urce] [Patch] [View Patch] [Gitweb] [Changelog
urce] [Patch] [View Patch] [View Inc.] [Gitweb] [Changelog

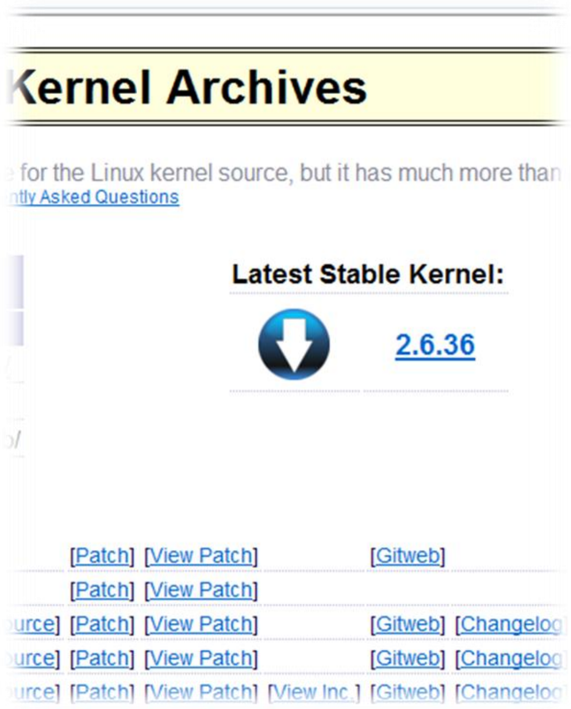
Linux-kernel mailing list FAQ
http://www.tux.org/lkml/

Kernel Version Numbers Contd.

Unless you are working on a “non-production” machine, it is recommended that you stay away from development kernels or kernels that are unstable or untested.

Kernel drivers interface directly with hardware. A mistake in a frequency setting or other setting can do physical damage to hardware components (i.e. melt memory cards or display adapters). Be very careful when changing the source code within the kernel!

Linux administrators should limit version numbers (or versions) of kernels in their environment at one time. The best administrators make sure all of their production machines have one version of kernel, and all of their development machines have a second, and finally all of their servers have a third. In this way, they only keep track of three kernel versions at one time. This minimal approach makes troubleshooting and updating easier.



Kernel Archives

for the Linux kernel source, but it has much more than

ntly Asked Questions

Latest Stable Kernel:

2.6.36

[Patch] [View Patch] [Gitweb]

[Patch] [View Patch]

urce] [Patch] [View Patch] [Gitweb] [Changelog

urce] [Patch] [View Patch] [Gitweb] [Changelog

urce] [Patch] [View Patch] [View Inc.] [Gitweb] [Changelog

Best Practices for Kernel Configuration

This is one method to use for kernel configuration. You will develop your own method as you become familiar with Linux processes.

1. The kernel should only contain necessary components. Nothing more. If you know what hardware is being used, what functionality is required, and what devices are going to be added, then you should configure the kernel with only those components. The kernel is a layer of system security, and the first rule of security is to give a user only enough functional to do his job—nothing more!
2. The kernel should be built and tested on a clean, up-to-date machine. Do not configure and compile a kernel on a machine that has been used for other purposes. You do not want to risk having unnecessary or unsafe code being compiled into a kernel as this will put the entire system at risk.
3. The kernel should always be installed from the official source tree at http://www.kernel.org or it should be installed based on a current version of the package (DEB or RPM) for your distribution. Always maintain version and release numbers for your new kernel.
4. Never delete the current kernel version when installing a new one before you know that the newly compiled version boots and provides the required functionality. It is very easy to end up with a non-working system. Always install the new kernel on a test machine prior to distributing it to production machines. Once you know everything works, then you may delete prior versions.

Compiling the Kernel (Overview)

Before the kernel becomes useable on a Linux system it must be compiled. This process takes human readable text (source code) and turns it into machine readable binary code (1's and 0's). Fortunately for us, the process is almost fully automated when we use a Linux system.

The process consists of a few steps prior to installation on the production machine:

1. Getting the source code
2. Installing the source
3. Configuring the kernel
4. Making the dependencies
5. Making the kernel image
6. Making the modules
7. Installing
8. Testing
9. Distributing

Depending on the build process, steps 4-6 may be reduced to a single step. The build (compiling) process may result in a kernel image file with modules, an RPM package, or a Debian package. For the remainder of this course, you are going to create each of these products.

RPM or Debian Package?

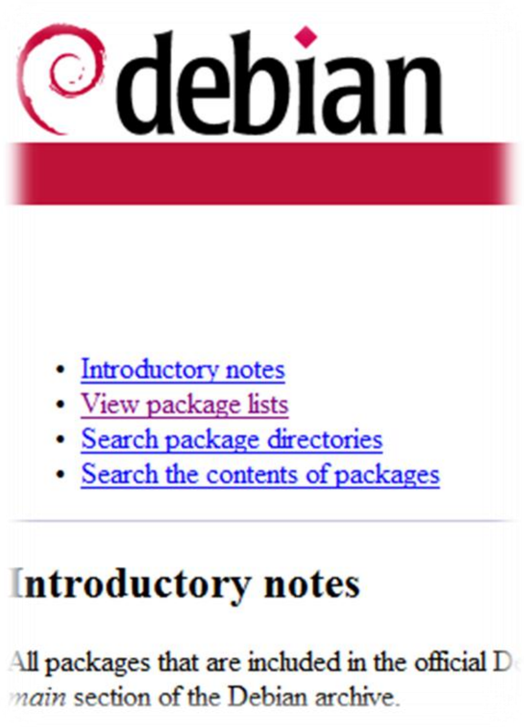
The Linux kernel can be built to an RPM or a Debian package, or it can be built and installed using the kernel image and modules. One of the tasks of the Linux Administrator is to decide which to use. The Linux kernel should never be distributed to end users as a *tar.bz2* file or as source files. When the distribution system is properly used, every file is accounted for and can be replaced or re-installed if files become corrupted.

Kernel Image and Modules:
Used on a development machine to test changes, and functionality. Can be used to identify the correct choices during the configuration process.

RPM Package:
Used to test, and release the newly compiled kernel to a Redhat based machine such as Redhat, Fedora, CentOS, and Mandriva.

DEB Package:
Used to test, and release the newly compiled kernel to a Debian based machine such as Debian, Ubuntu, Kubuntu, and Linspire.

It is good practice not to release software using the default kernel image files and dependencies.



debian

- Introductory notes
- View package lists
- Search package directories
- Search the contents of packages

Introductory notes

All packages that are included in the official D
main section of the Debian archive.

Recommended Reading
Linux Kernel
RPM Package

Updating Existing Debian Installation

Before starting software development, you must first update your build system to the latest version of software. Then, create the build environment and install all necessary dependencies.
Note: Press **Enter** on your keyboard after each command (in bold).

1. Log into your Debian-based system under your normal user account.
2. Go to **Applications →Accessories → Terminal** from the menu.
3. At a command prompt type **sudo apt-get update**
4. Enter your password when requested and the system will update all package databases.
5. Once back at the command prompt type **sudo apt-get upgrade**
6. If requested for your password, enter it.
7. If prompted to accept the changes press **Y** (for yes)
8. Allow the system to update. This may take some time depending on the amount of software that needs to be installed.
9. Install the build scripts using the following command:
   sudo apt-get install build-essential fakeroot kernel-package linux-source libncurses5-dev bzip2 wget
10. Install the suggested packages. Note: you may skip the documentation packages ending in **.doc** by cutting and pasting or typing them into an **apt-get install** line.
11. Add and delete packages as requested to resolve any errors during the installation process.
12. Do a final **sudo apt-get upgrade** when complete.

Select **PLAY** below to view a video on updating an existing Debian-based installation.

View Video
VideoLesson9UpdateDebianInstall(C2L9S13).mp4

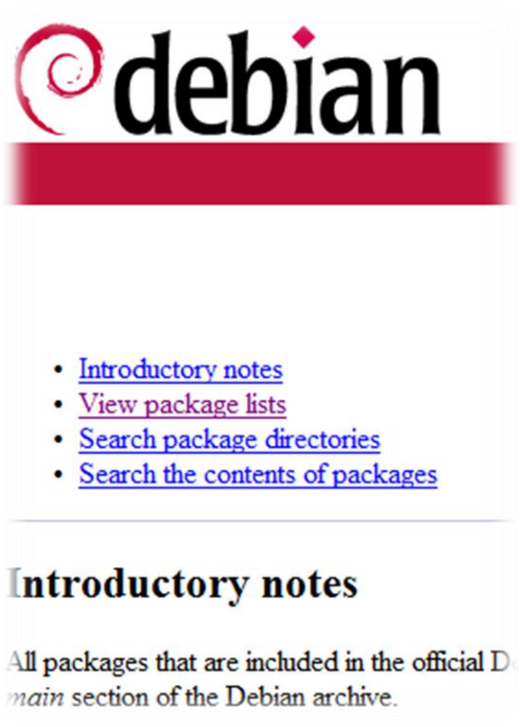
Important:
Continue to next slide for remaining steps (13-23).

Updating Existing Debian Installation

Note: Press **Enter** on your keyboard after each command (in bold).

13. Add user to src group by typing **sudo adduser cmolnar src**
14. Reboot to pick up the new groups and packages
15. Open the terminal window again.
16. Change to the /usr/src directory by typing **cd /usr/src**
17. Use FTP to get the latest source code from the ftp.kernel.org site. It is in the /pub/linux/kernel directory. Type **ftp –p** ftp.kernel.org
18. Type **cd pub/linux/kernel/v2.6**
19. Type **ls linux*.bz2** to find the most recent version of kernel. Look for the most recent file.
20. Type **get *latestkernelfilename*** to start download.
21. Once file received type **quit**
22. Type **tar xvjf *latestkernelfilename*** to un-archive the kernel source code.
23. Create a link to the kernel source code by typing **ln –s ./linux-2.6.xx linux** (Replace xx with kernel number).

At this point the system is configured, and the Linux kernel source code has been installed.


debian
Introductory notes
View package lists
Search package directories
Search the contents of packages
Introductory notes
All packages that are included in the official D
main section of the Debian archive.

Recommended Reading

Tar manpage

Ftp manpage

Configuring the Kernel

After updating the build system and installing the kernel source, your next step is configuring the kernel.

The program to configure the kernel is built directly into the kernel source package. There are two ways to do this configuration: (1) use a graphical, mouse-driven interface, or (2) use a terminal based menu driven interface. In either circumstances, begin with a kernel configuration that works—preferably, the same kernel running on the build machine.

1. From the terminal window, access the Linux source directory by typing **cd /usr/src/linux/**
2. Type **ls /boot/config*** to list the installed configuration versions. The **uname –r** command will help you identify the correct version.
3. Copy the configuration file to your Linux source directory by typing, **cp /boot/config-`uname –r` ./.config**
4. Verify the configuration file has been copied to your source directory by typing **ls –la .config** (You see a single file with today's date).
5. Type **make menuconfig** to begin configuration process. If it errors out, look at the message. You may need to change your terminal screen to a larger setting.
6. Use the menu options to make necessary configuration changes.
7. Use the tab key to jump to the Exit button. **Exit.**
8. Select **Yes** when asked to save the file. (Command prompt displays)

Select **PLAY** below to view the Debian kernel configuration video.

View Video VideoLesson9ConfiguringKernel(C2L9S15).mp4

Required Reading:
Kernel configuration
Uname manual page

Building the Kernel

Now that you have installed and configured the kernel source, the next step in the process is to build the kernel package.

1. From the terminal window, enter the Linux source directory by typing, **cd /usr/src/linux/**
2. Type **make-kpkg clean** to clean the kernel build files. This command gets rid of old files that may have been distributed in error.
3. Type **fakeroot make-kpkg –initrd –append-to-version=-customkernel kernel_image kernel_headers** to initiate the kernel build process.
4. The kernel build may take 2-3 hours depending on the size of the kernel, your configuration, processor speed, and other activities. Try not to do anything else on the machine while the kernel is building.
5. A command prompt (without error messages) will appear when complete.
6. Type **cd /usr/src**
7. Type **ls –l linux*.deb** to view your new Debian packages in that directory.
8. Install using **dpkg –install *dpkgname***
9. Make sure you have a boot disk or CD in case you need to troubleshoot.
10. Reboot the system with the new kernel image using **sudo shutdown –r 0**
11. If asked on the reboot, select your new kernel from the menu.

If you have any problems with your system when you reboot, use your keyboard to get to a terminal window and correct the changes. To correct errors, you may have to recompile the kernel, or uninstall your new kernel and allow your system to reboot using the original kernel.

Select **PLAY** below to view a video on building a Debian-based kernel.

View Video VideoLesson9BuildingKernel(C2L9S16).mp4

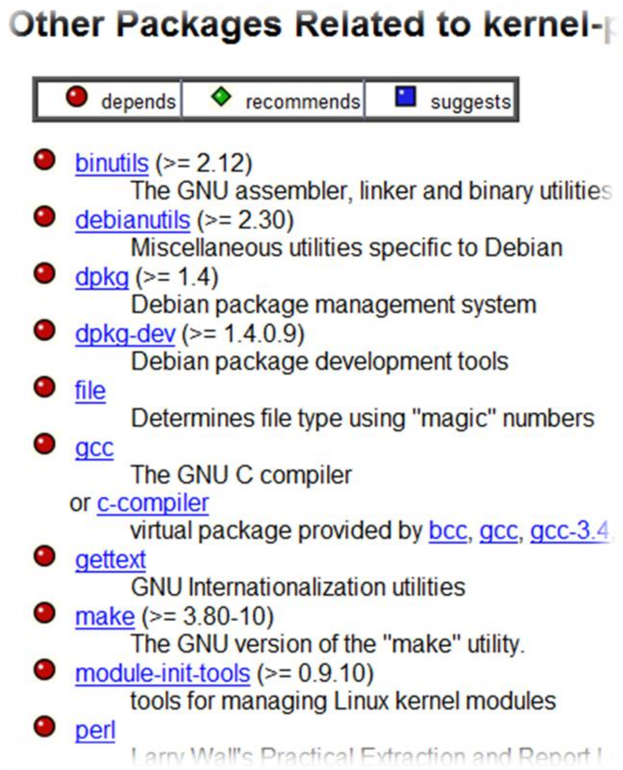
Required Reading:
Kernel packaging on Debian

Summary: Debian Kernel Package

To summarize the build process:

1. Update the build system.
2. Setup the build directories.
3. Install the build software.
4. Get the source packages.
5. Un-archive the kernel package
6. Copy configuration file to build directory.
7. Run configuration utility with make.
8. Save configuration file.
9. Run **make deb-pkg** to compile and build packages.
10. Install using dpkg when complete onto a test machine.
11. Fix any errors.
12. Distribute.

This is one of many ways to build the kernel. For Debian-based machines, I choose the default kernel build process. Each flavor (distribution) has a slightly different process; this process will work on all machines. The process is a little different on RPM based machines.



Other Packages Related to kernel-

● depends ◆ recommends ■ suggests

- ● binutils (>= 2.12)
  The GNU assembler, linker and binary utilities
- ● debianutils (>= 2.30)
  Miscellaneous utilities specific to Debian
- ● dpkg (>= 1.4)
  Debian package management system
- ● dpkg-dev (>= 1.4.0.9)
  Debian package development tools
- ● file
  Determines file type using "magic" numbers
- ● gcc
  The GNU C compiler
  or c-compiler
  virtual package provided by bcc, gcc, gcc-3.4
- ● gettext
  GNU Internationalization utilities
- ● make (>= 3.80-10)
  The GNU version of the "make" utility.
- ● module-init-tools (>= 0.9.10)
  tools for managing Linux kernel modules
- ● perl
  Larry Wall's Practical Extraction and Report

Required Reading:
Linux Kernel makefile

RPM Packaging

Creating RPM Build Environment

As with Debian, before you can build any software on a Redhat system, you need to create the build environment and install the build tools for your distribution. In this task, you will learn about the RPM build environment. The steps to creating this environment are demonstrated in the video to your right. These steps are:

Note: Press the **Enter** key on your keyboard after each command (in bold).

1. Log in under your regular user id.
2. Type **sudo yum upgrade** to update your build system to the latest version.
3. Type **sudo yum install rpmdevtools** to install the build software.
4. Type **sudo yum groupinstall "Development Tools"** to install all of the development tools.
5. Type **sudo yum install rpm-build** to install the build scripts.
6. Next we need to create the build directory structure.
7. Type **cd** to make sure you are in your home directory.
8. Type **rpmdev-setuptree** to setup the Redhat build tree.
9. Type **cd** to return to your home directory.
10. Type **yumdownloader –source kernel** to download the kernel source.
11. Type **ls** to locate the src.rpm file that was just downloaded.
12. Type **sudo yum-builddep filename.src.rpm**
13. Note: The filename after yum-builddep is the file name from the step above.
14. Allow the system to install all additional packages that it wishes to install.
15. Type **rpm –Uvh filename.src.rpm** to install the kernel source.
16. Note: the filename is the src.rpm file from step 11.

Select **PLAY** below to view the RPM build video.

View Video VideoLesson9CreatingRPMEnvironment(C2L9S19).mp4

Required Reading:
Building kernel on Fedora
Fedora kernel build process

Configure the Kernel for RPM Packaging

The same rules apply to the configuration in the RPM version as the DEB version. Do not include more than you need to, make sure security is taken into account, and keep the size of the end kernel small.

1. Type **cd ~/rpmbuild/SPECS** to change into the build SPECS directory.
2. Type **rpmbuild –bp –target=\`uname –m\` kernel.spec** to unpack the source code package.
3. Type **cd ~/rpmbuild/BUILD/kernel-2.6.*$ver*/linux-2.6.*$ver.$arch*** where the *$ver* is the version number of the kernel and the *$arch* is the architecture of your machine. You can find the architecture of your machine by typing **uname –m** at a command prompt.
4. Start with your default configuration file by copying the one from your boot directory. Use the command **cp /boot/config-\`uname –r\` ./.config** to do this.
5. Type **make menuconfig** to begin the menu configuration.
6. Make the necessary changes to the configuration.
7. Select **Exit** and then **Save configuration** when complete.
8. Use a text editor, such as vi or pico, and add the architecture of the machine at the top of the file in the following form: **# x86_64** for a 64 bit Intel 86 machine. You can find your architecture by typing **uname –i** at the command prompt.
9. Type **cp .config ~/rpmbuild/SOURCES/config-*$arch*-generic** to copy the configuration file to the sources directory so the rpm build scripts can find it.

Your new kernel is configured and ready to build.

Select **PLAY** below to view a video on configuring a kernel for RPM packaging.

View Video VideoLesson9ConfigureKernelRPMPackaging(C2L9S20) mp4

Required Reading:
RPM build commands
RPM build manpage

Building the Kernel for RPM Packaging

Since you have your source package installed from the last step and have made your changes, it is time to build the new installable packages.

1. Type **cd ~/rpmbuild/SPECS** to change to the build specs directory.
2. Use your editor (vi or pico) to open the kernel.spec file. For **vi** the command is **vi kernel.spec**
3. Locate the entry for **define buildid**
4. In vi type: **/define buildid** and the cursor would be placed at the text.
5. Add a version identifier after the *build id* to identify your version of kernel. It must start with a . (period) and cannot have spaces.
6. Save the file. In vi your command would be **:w**
7. Exit the editor. In vi your command would be **:q**
8. To compile the kernel, type **rpmbuild -bb --with baseonly --without debug info --target=`uname -m` kernel.spec**
9. This process may take a few hours.
10. When complete, change to your package directory by typing: **cd ~/rpmbuild/RPMS/**
11. You can install your new kernel by using the **yum –nogpgcheck localinstall package.rpm** command.

Install the new kernel on a test machine without before releasing to other machines. You may need to rebuild once or twice to work out any bugs or configuration problems. Try not to install anything on your build machine unless you are prepared to re-install it.

Select **PLAY** below to view a video on building a kernel for RPM packaging.

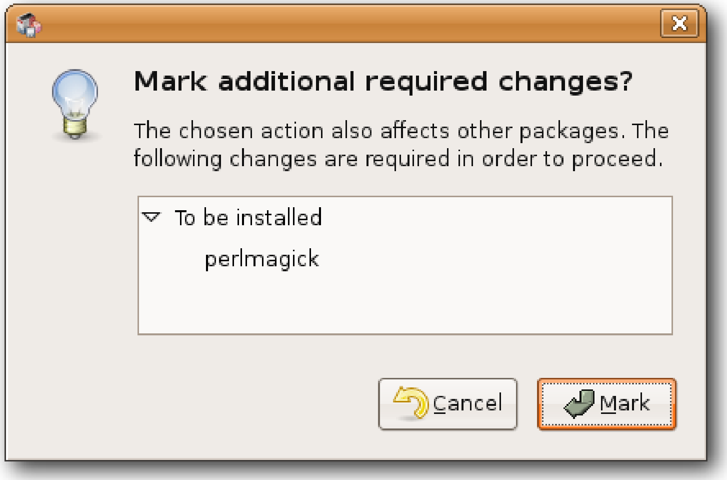
View Video VideoLesson9BuildKernelRPMPackaging(C2L9S21). mp4

Required Reading:
RPM spec file
RPM build process

Summary of RPM Packaging

To summarize the RPM build process:

1. Update the build system
2. Setup the build directories
3. Install the build software
4. Get the source packages
5. Install the source packages
6. Make changes to the configuration file
7. Build the new kernel
8. Test install using **sudo yum –nogpgcheck localinstall** to make sure it installs cleanly on a test machine
9. Test all functionality
10. Distribute


Mark additional required changes?

The chosen action also affects other packages. The following changes are required in order to proceed.

To be installed
perlmagick

Manual Kernel Build

Configure for Manual Kernel Build

The first step to building a kernel manually is to make sure that your machine can compile either an RPM or a Debian kernel package based on your machine type. Once you know it is able to do that you can follow these steps:

Note: Press the Enter key after each command line entry (in bold).

1. Enter the terminal for your system and make it to full screen.
2. Use your software management system to update your machine to the latest version of software. Example: **sudo yum update** or **sudo apt-get upgrade**
3. Create a *code* directory in your home directory by typing **mkdir ~/code**
4. Change into the code directory by typing **cd ~/code**
5. Use FTP to get the latest source code from the ftp.kernel.org site. It is in the /pub/linux/kernel directory. Type **ftp –p ftp.kernel.org**
6. Type **cd pub/linux/kernel/v2.6**
7. Type **ls linux*.bz2** (Identify the most recent file version of the kernel).
8. Type **get *latestkernelfilename*** to start download.
9. Once the file has been received type **quit**
10. Type **tar xvjf *latestkernelfilename*** to un-archive the kernel source code.
11. Create a link to the kernel source code by typing **ln –s ./linux-2.6.xx linux** (Replace xx with the kernel version number).

The system is now configured and the Linux kernel source code installed. You may now apply patch files or changes.

Select **PLAY** below to view a video on configuring a manual kernel build.

View Video VideoLesson9ConfigureManualKernelBuild(C2L9S24). mp4

Patch Files

The company for which you work is running Linux computer systems. They purchased a new piece of computer equipment that came with drivers for Microsoft Windows and Apple OS/X.

The company's internal development team modified the Linux kernel to provide the functionality for the new hardware. They gave you an archive with the full kernel code and told you to distribute to over 40 Linux desktops in the company. From your training, you know you need to use a distribution package such as RPM or DEB to do this, but it is impossible to create this distribution from the archive the developers gave you.

Every Linux install contains a utility called "diff." The diff utility compares two files (or directories) and outputs the result in a machine parsable text file. In the above scenario, we can use the diff utility to compare the changed code to the original kernel package and then include the resulting patch file in the build process.

The first step is to create the patch file using the diff command.

Required Reading:
Diff and Patch
How to create patch files

Create Patch Files

The following is the procedure to create the patch file.

1. From the terminal window type **cd ~/code/**
2. In this directory you should have downloaded the kernel source archive. If you have not done so, please return to C2L9S29 (two previous slides) and follow the directions.
3. Make sure the only file in this directory is linux-2.6.xx.tar.bz2 (xx is the release number).
4. Use the tar utility to un-archive the original distribution file by typing **tar xvjf linux-2.6.xx.tar.bz2** at the command prompt.
5. Copy the un-archived file to a new directory by typing **cp –ar linux-2.6.xx linux-2.6.xx-new** (replace xx with the correct release numbers).
6. If you were given a patched directory (as in our scenario above) you would not copy the original source. Instead, you would put the patched directory into the code directory and skip the next two steps, since the changes have already been made.
7. Change into the new directory by typing **cd linux-2.6.xx-new**
8. Make any changes that you wish to make. For practice purposes in the video lesson, we are adding a file and changing a couple lines in other files.
9. Once completed, change back into the code directory by typing **cd ~/code**
10. Use the diff command to create the patch file by typing **diff –r –c --new-file linux-2.6.xx linux-2.6.xx-new > my.patch**

Select **PLAY** below to view a video on patch files.

View Video VideoLesson9CreatePatchFiles(C2L9S26). mp4

Required Reading:
Diff man page

Use the Patch File

To check the patch file you created in the last slide, you will use the patch command.

1. Change to the *~/code* directory by typing **cd ~/code**
2. Remove all files from the code directory except the original *linux-2.6.xx.tar.bz2* and the *my.patch* file. Remember you can use the **rm filename** command to remove files. Use **rm –rf directoryname** to remove directories.
3. Un-archive the original archive file using **tar xvjf linux-2.6.xx.tar.bz2** command. (Replace xx with the release numbers).
4. Change to the *linux-2.6.xx* directory by typing **cd linux-2.6.xx**
5. Apply the patch file using the patch command.
6. Type **patch –p1 –i ../my.patch**
7. You will get a list of files changed, and a confirmation the patch was successfully applied. If you get errors, try the patch again from a fresh directory after fixing errors.
8. Once the patch is successfully applied, verify that the patch worked.
9. Verify the patch by checking the contents of the files you changed, or the files affected by the patch.

Now that you have the patch file, include it into the *rpmbuild/SOURCES* directory of your RPM-based system, or into the build scripts of the Debian system. Remember to put into the **SPECS/kernel.spec** file in an RPM build system in the patches section.

Select **PLAY** below to view a video on using patch files.

View Video VideoLesson9UsePatchFiles(C2L9S27). mp4

Required Reading:
Patch man page

Manually Configuring the Kernel

The next step in the manual build process is to configure the kernel. In this example, you will use the menu configuration since it works in terminal mode.

Note: remember to press [**ENTER**] after every command line entry.

1. From the terminal window access the Linux source directory by typing, **cd /usr/src/linux/**
2. Type **ls /boot/config*** to list the installed configuration versions. Don't worry, the **uname –r** command will help you pick the correct version.
3. Copy the configuration file to your Linux source directory by typing, **cp /boot/config-`uname –r` ./.config**
4. Verify that the file is there by typing **ls –la .config** (you should see a single file with today's date).
5. Type **make menuconfig** to begin the configuration process. If it errors out, examine the message. You may need to change your terminal screen to a larger setting.
6. Using the menu options, change the configuration settings you wish to change.
7. When complete, tab to the bottom of the menu screen and **Exit**.
8. Select **Yes** when asked to save the file.

You will return to a command prompt.

Select **PLAY** below to view a video on manually configuring the kernel.

View Video VideoLesson9ManuallyConfigureKernel(C2L9S28).mp4

Manually Building the Kernel

The next step in the manual build process is to configure the kernel. In this example, you will use the menu configuration since it works in terminal mode.

Note: remember to press [**ENTER**] after every command line entry.

1. To manually build the kernel you want to begin in the Linux source directory. Type **cd ~/code/linux** to get there.
2. Use the *make* command to build the kernel. Type **make all**
3. The compiler will take 2-3 hours to complete this task.
4. When completed, the command prompt will display. If you get errors, fix them, or ask a developer for help.

We are not going to install the resulting files used in this example because it is bad practice to do so. You can save your configuration file *.config* and use it for one of the package build processes already described in this lesson.

Select **PLAY** below to view a video on manually configuring the kernel.

View Video VideoLesson9ManuallyBuildKernel(C2L9S29).mp4

Required Reading:
Building and installing Linux kernel

Lesson Summary

Building kernel packages is one of the most important functions of the Linux administrator. While a kernel can be built for one machine without using the package management system, it is not considered good kernel or package management to do so. The package management system allows us to track changes, install, and uninstall software. With kernel packages, the uninstall function is of great importance in case of errors in the build process.

In this lesson you also explored the build process for a kernel without a package manager that allows you to test your changes and perhaps test code changes given to you by a developer. Kernel builds should not be used in a production environment.

All software installed on a Linux machine should be built into packages. Generally, administrators who update kernels without a package management system are not respected or well-regarded.