RH033 - Red Hat Linux Essentials

Introduction

RH033: Red Hat Linux Essentials

RH033-RHEL5u4-en-8-20090923/7940b128title

Copyright

Welcome

Please let us know if you need any special assistance while visiting our training facility.

Please introduce yourself to the rest of the class!

Red Hat Enterprise Linux

- Enterprise-targeted Linux operating system
- Focused on mature open source technology
- Extended release cycle between major versions
  - With periodic minor releases during the cycle
  - Certified with leading OEM and ISV products
- All variants based on the same code
  - Certify once, run any application/anywhere/anytime
- Services provided on subscription basis

Red Hat Enterprise Linux Variants

- Red Hat Enterprise Linux Advanced Platform
  - Unlimited server size and virtualization support
  - HA clusters and cluster file system
- Red Hat Enterprise Linux
  - Basic server solution for smaller non-mission-critical servers
  - Virtualization support included
- Red Hat Enterprise Linux Desktop
  - Productivity desktop environment
  - *Workstation* option adds tools for software and network service development
  - *Multi-OS* option for virtualization

Red Hat Subscription Model

- Red Hat sells subscriptions that entitle systems to receive a set of services that support open source software
    - Red Hat Enterprise Linux and other Red Hat/JBoss solutions and applications
- Customers are charged an annual subscription fee per system
    - Subscriptions can be migrated as hardware is replaced
    - Can freely move between major revisions, up and down
    - Multi-year subscriptions are available
- A typical service subscription includes:
    - Software updates and upgrades through Red Hat Network
    - Technical support (web and phone)
    - Certifications, stable APIs/versions, and more

Contacting Technical Support

- Collect information needed by technical support:
  - Define the problem
  - Gather background information
  - Gather relevant diagnostic information, if possible
  - Determine the severity level
- Contacting technical support by WWW:
  - http://www.redhat.com/support/
- Contacting technical support by phone:
  - See http://www.redhat.com/support/policy/sla/contact/
  - US/Canada: 888-GO-REDHAT (888-467-3342)

Red Hat Network

- A systems management platform providing lifecycle management of the operating system and applications
  - Installing and provisioning new systems
  - Updating systems
  - Managing configuration files
  - Monitoring performance
  - Redeploying systems for a new purpose
- "Hosted" and "Satellite" deployment architectures

Red Hat Services and Products

- Red Hat supports software products and services beyond Red Hat Enterprise Linux
  - JBoss Enterprise Middleware
  - Systems and Identity Management
  - Infrastructure products and distributed computing
  - Training, consulting, and extended support
- http://www.redhat.com/products/

Fedora and EPEL

- Open source projects sponsored by Red Hat
- Fedora distribution is focused on latest open source technology
  - Rapid six month release cycle
  - Available as free download from the Internet
- EPEL provides add-on software for Red Hat Enterprise Linux
- Open, community-supported proving grounds for technologies which may be used in upcoming enterprise products
- Red Hat does not provide formal support

Objectives

- A user who can use effectively employ Red Hat Enterprise Linux to customize his or her operating environment as well as accomplish common command-line tasks and desktop productivity roles

Audience and Prerequisites

- Audience: Users new to Linux and UNIX; users and administrators transitioning from another operating system
- User-level experience with any computer system; use of mouse, menus and any graphical user interface

Pre/Post-Assessments

- Some units begin with a pre-assessment
    - 3-5 simple questions about the unit's subject
    - Just leave blank if you don't know the answer
- Questions are asked again at the end of the unit

Lab Exercises

- Labs
  - Fundamental exercise providing basic goals, reinforcing the lecture
- Lab Solutions
  - Offers step-by-step detailed methodology
  - Found for all exercises that do not have specific steps themselves
- Challenge Labs
  - Advanced exercise, reinforcing more advanced topics from the lecture
  - Not all students may have the time to complete
- Optional Labs
  - Optional exercise that may depend on classroom specific environment

Classroom Network

- example.com network (192.168.0.0/24)
  - instructor.example.com (192.168.0.254)
    - Main classroom server: Provides DHCP, DNS, routing and other services
  - station*X*.example.com (192.168.0.*X*)
    - Student systems
  - server*X*.example.com (192.168.0.*X+100*)
    - Virtual server hosted on student stations (Not used in all classes)
- remote.test network (192.168.1.0/24)
  - cracker*X*.remote.test (192.168.1.*X*)
    - Virtual client hosted on student systems (Not used in all classes)

Notes on Internationalization

- Red Hat Enterprise Linux supports nineteen languages
- Default system-wide language can be selected
  - During installation
  - With **system-config-language** (System->Administration->Language)
- Users can set personal language preferences
  - From graphical login screen (stored in `~/.dmrc`)
  - For interactive shell (with `LANG` environment variable in `~/.bashrc`)
  - Alternate languages can be used on a per-command basis:

```
[user@host ~]$ LANG=ja_JP.UTF-8 date
```

Lecture 1

Linux Ideas and History

Objectives

Upon completion of this unit, you should be able to:

- Explain the nature of open source software
- Discuss the origins of Linux
- List the Red Hat operating system distributions
- Explain basic Linux principles

What is Open Source?

- Open source: software and source code available to all
- The Free Software Foundation specifies four freedoms
  - The freedom to run the program for any purpose.
  - The freedom to study and modify the source code
  - The freedom to redistribute the program
  - The freedom to create derivative programs
- Many open-source licenses exist, each with different particulars

Supplemental Media

Fedora developer Jeremy Katz on the advantages of open source

Linux Origins

- 1984: The GNU Project and the Free Software Foundation
  - Creates open source version of UNIX utilities
  - Creates the General Public License (GPL)
    - Software license enforcing open source principles
- 1991: Linus Torvalds
  - Creates open source, UNIX-like kernel, released under the GPL
  - Ports some GNU utilities, solicits assistance online
- Today:
  - Linux kernel + GNU utilities = complete, open source, UNIX-like operating system
    - Packaged for targeted audiences as *distributions*

Supplemental Media

Linus Torvalds on how to pronounce "Linux"

Red Hat Distributions

- Linux *distributions* are OSes based on the Linux kernel
- Red Hat Enterprise Linux
  - Stable, thoroughly tested software
  - Professional support services
  - Centralized management tools for large networks
- The Fedora Project
  - More, newer applications
  - Community supported (no official Red Hat support)
  - For personal systems

Supplemental Media

Fedora developer Jeremy Katz on the relationship between Red Hat Enterprise Linux and Fedora

Linux principles

- Everything is a file (including hardware)
- Small, single-purpose programs
- Ability to chain programs together to perform complex tasks
- Avoid captive user interfaces
- Configuration data stored in text

End of Lecture 1

- Questions and Answers
- Summary
  - Open source and the right to modify
  - The GNU Project and the Free Software Foundation
  - Linus Torvalds and the Linux kernel
  - Red Hat Enterprise Linux and the Fedora Project
  - Basic Linux Principles

Lecture 2

Linux Usage Basics

Objectives

Upon completion of this unit, you should be able to:

- Log into a Red Hat Enterprise Linux system
- Start X from a console
- Access the command line from X
- Change your password
- Understand the nature of root privileges
- Elevate your privileges
- Edit plain text files

Logging in to a Linux System

- Login using username and password
- Two types of login screens: text-based and graphical
    - Text-based login leaves you at a *shell prompt*
    - Graphical login starts a *desktop environment*
- Each user has a *home directory* for personal file storage
    - User-specific configuration data is often kept there as well

Switching between virtual consoles and the graphical environment

- A typical Linux system will run six virtual consoles and one graphical console
  - Server systems often have only virtual consoles
  - Desktops and workstations typically have both
- If graphical console is inactive, it may be started manually
  - The X server must be pre-configured by the system administrator
  - Log into a virtual console and run **startx**
- Switch among virtual consoles by typing: **Ctrl-Alt-F*[1-6]***
- Access the graphical console by typing **Ctrl-Alt-F7**

gnome-terminal

- Applications->Accessories->Terminal
- Graphical terminal emulator that supports multiple "tabbed" shells
  - **Ctrl**-**Shift**-**t** creates a new tab
  - **Ctrl**-**PgUp/PgDn** switches to next/prev tab
  - **Ctrl**-**Shift**-**c** copies selected text
  - **Ctrl**-**Shift**-**v** pastes text to the prompt
  - **Shift**-**PgUp/PgDn** scrolls up and down a screen at a time

Changing Your Password

- Passwords control access to the system
- General guidelines for best security:
  - Change the password the first time you log in
  - Change it regularly thereafter
  - Select a password that is hard to guess
- To change your password:
  - GUI: System->Preferences->About Me and then click Change Password
  - CLI: **passwd**

The *root* user

- The *root* user: a special administrative account
  - Also called the *superuser*
  - `root` has near complete control over the system
    - ...and a nearly unlimited capacity to damage it!
- Do not login as `root` unless necessary
  - Normal (*unprivileged*) users' potential to do damage is more limited

Changing Identities

- **su -** creates new shell as root
- **sudo *command*** runs ***command*** as root
  - Requires prior configuration by a system-administrator
- **id** shows information on the current user

Command Line Shortcuts
The Tab Key

- Type **Tab** to complete command lines:
  - For the command name, it will complete a command name
  - For an argument, it will complete a file name
- Examples:

```
$ xte<Tab>
$ xterm
$ ls myf<Tab>
$ ls myfile.txt
```

Command Line Shortcuts
History

- **bash** stores a history of commands executed
- **history** lists all commands
- **history *N*** lists the last *N* commands

```
$ history 5
14  cd /tmp
15  ls -l
16  cd
17  cp /etc/passwd .
18  vi passwd
```

More History Tricks

- Use the **up** and **down** keys to scroll through previous commands
- Type **Ctrl-r** to search for a command in command history.
  - *(reverse-i-search)`':*
- To recall last argument from previous command:
  - **Esc,.** (the escape key followed by a period)
  - **Alt-.** (hold down the alt key while pressing the period)
  - Can be pressed multiple times
  - **!$** (only valid for the last command)

Editing text files

- The **nano** editor
  - Easy to learn, easy to use
  - Not as feature-packed as some advanced editors
- Other editors:
  - **gedit**, a simple graphical editor
  - **vim**, an advanced, full feature editor
  - **gvim**, a graphical version of the vim editor

End of Lecture 2

- Questions and Answers
- Summary
  - Login name and password
  - **startx**
  - **gnome-terminal**
  - **passwd**
  - **su**
  - **nano**

Lecture 3

Running Commands and Getting Help

Objectives

Upon completion of this unit, you should be able to:

- Execute commands at the prompt
- Explain the purpose and usage of some simple commands
- Use the built-in help resources in Red Hat Enterprise Linux

Running Commands

- Commands have the following syntax:
  - **`command`** ***`options arguments`***
- Each item is separated by a space
- Options modify a command's behavior
  - Single-letter options usually preceded by **`-`**
    - Can be passed as **`-a -b -c`** or **`-abc`**
  - Full-word options usually preceded by **`--`**
    - Example: **`--help`**
- Arguments are file names or other data needed by the command
- Multiple commands can be separated by **;**

Some Simple Commands

- **date** - display date and time
- **cal** - display calendar

Getting Help

- Do not try to memorize everything!
- Many levels of help
  - **whatis**
  - ***command*** **--help**
  - **man** and **info**
  - `/usr/share/doc/`
  - Red Hat documentation

The whatis Command

- Displays short descriptions of commands
- Uses a database that is updated nightly
- Often not available immediately after install

```
$ whatis cal
cal      (1)  - displays a calendar
```

The --help Option

- Displays usage summary and argument list
- Used by most, but not all, commands

```
$ date --help
Usage: date [OPTION]... [+FORMAT] or:
date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
Display the current time in the given FORMAT,
or set the system date.
...argument list omitted...
```

Reading Usage Summaries

- Printed by **--help**, **man** and others
- Used to describe the syntax of a command
  - Arguments in [] are optional
  - Arguments in CAPS or <> are variables
  - Text followed by ... represents a list
  - x|y|z means "x or y or z"
  - -abc means "any mix of -a, -b or -c"

The man Command

- Provides documentation for commands
- Almost every command has a man "page"
- Pages are grouped into "chapters"
- Collectively referred to as the Linux Manual
- **man [<chapter>] <command>**

Navigating man Pages

- While viewing a man page
  - Navigate with arrows, **PgUp**, **PgDn**
  - ***/text*** searches for text
  - **n**/**N** goes to next/previous match
  - **q** quits
- Searching the Manual
  - **man -k *keyword*** lists all matching pages
  - Uses **whatis** database

The info Command

- Similar to **man**, but often more in-depth
- Run **info** without args to list all page
- **info** pages are structured like a web site
  - Each page is divided into "nodes"
  - Links to nodes are preceded by *
  - **info [*command*]**

Navigating info Pages

- While viewing an **info** page
  - Navigate with arrows, **PgUp**, **PgDn**
  - **Tab** moves to next link
  - **Enter** follows the selected link
  - **n**/**p** /**u**/**l** goes to the next/previous/up-one/last node
  - **s** ***text*** searches for text (default: last search)
  - **q** quits **info**

Extended Documentation

- The `/usr/share/doc` directory
  - Subdirectories for most installed packages
  - Location of docs that do not fit elsewhere
    - Example configuration files
    - HTML/PDF/PS documentation
    - License details

Red Hat Documentation

- Available at http://www.redhat.com/docs/
  - Installation Guide
  - Deployment Guide
  - Virtualization Guide
- Knowledge base: http://kbase.redhat.com/
  - Common questions and their solutions
- Deployment Guide
  - System->Documentation->Deployment Guide
  - **yelp ghelp:Deployment_Guide**

End of Lecture 3

- Questions and Answers
- Summary
  - Running Commands
  - Getting Help

Lecture 4

Browsing the Filesystem

RH033-RHEL5u4-en-8-20090923/a31efc43title

Objectives

Upon completion of this unit, you should be able to:

- Describe important elements of the filesystem hierarchy
- Copy, move, and remove files
- Create and view files
- Manage files with Nautilus

Linux File Hierarchy Concepts

- Files and directories are organized into a single-rooted inverted tree structure
- Filesystem begins at the *root* directory, represented by / (forward slash).
- Names are case-sensitive
- Paths are delimited by /

Some Important Directories

- Home Directories: `/root`,`/home/`*`username`*
- User Executable: `/bin`, `/usr/bin`, `/usr/local/bin`
- System Executables: `/sbin`, `/usr/sbin`, `/usr/local/sbin`
- Other Mountpoints: `/media`, `/mnt`
- Configuration: `/etc`
- Temporary Files: `/tmp`
- Kernels and Bootloader: `/boot`
- Server Data: `/var`, `/srv`
- System Information: `/proc`, `/sys`
- Shared Libraries: `/lib`, `/usr/lib`, `/usr/local/lib`

File and Directory Names

- Names may be up to 255 characters
- All characters are valid, except the forward-slash
  - It may be unwise to use certain special characters in file or directory names
  - Some characters should be protected with quotes when referencing them
- Names are case-sensitive
  - Example: `MAIL`, `Mail`, `mail`, and `mAiL`
  - Again, possible, but may not be wise

Using Nautilus

- Gnome graphical filesystem browser
- Can run in *spatial* or *browser* mode
- Accessed via...
  - Desktop icons
    - Home: Your home directory
    - Computer: Root filesystem, network resources and removable media
  - Applications->System Tools->File Browser

Moving and Copying in Nautilus

- Drag-and-Drop
  - Drag: Move on same filesystem, copy on different filesystem
  - Drag + **Ctrl**: Always copy
  - Drag + **Alt**: Ask whether to copy, move or create symbolic link (alias)
- Context menu
  - Right-click to rename, cut, copy or paste

File Management from the Command-Line

- Shells typically start in the home directory
- Change directory with **cd**
- List directory contents with **ls**
- Manage files with **cp**, **mv** and **rm**

Determining your Current Directory

- Each shell and system process has a *current working directory*(cwd)
- **pwd**
  - Displays the absolute path to the shell's cwd

Absolute and Relative Pathnames

- Used when referring to files on the command-line
- Absolute pathnames
  - Begin with a forward slash
  - Complete "road map" to file location
  - Can be used anytime you wish to specify a file name
- Relative pathnames
  - Do not begin with a slash
  - Specify location relative to your current working directory
  - Can be used as a shorter way to specify a file name

Changing Directories

- **cd** changes directories
  - To an absolute or relative path:
    - **cd** `/home/joshua/work`
    - **cd** `project/docs`
  - To a directory one level up:
    - **cd** `..`
  - To your home directory:
    - **cd**
  - To your previous working directory:
    - **cd** `-`

Listing Directory Contents

- Lists the contents of the current directory or a specified directory
- Usage:
  - **ls [options] [*files_or_dirs*]**
- Example:
  - **ls -a** (include hidden files)
  - **ls -l** (display extra information)
  - **ls -R** (recurse through directories)
  - **ls -ld** (directory and symlink information)

Copying Files and Directories

- **cp** - copy files and directories
- Usage:
  - **cp [options]** ***file destination***
- More than one file may be copied at a time if the destination is a directory:
  - **cp [options]** ***file1 file2 destdir***

Copying Files and Directories: The Destination

- If the destination is a directory, the copy is placed there
- If the destination is a file, the copy overwrites the destination
- If the destination does not exist, the copy is renamed

Moving and Renaming Files and Directories

- **mv** - move and/or rename files and directories
- Usage:
  - **mv [options]** ***file destination***
- More than one file may be moved at a time if the destination is a directory:
  - **mv [options]** ***file1 file2 destdir***
- Destination works like **cp**

Creating and Removing Files

- **touch** - create empty files or update file timestamps
- **rm** - remove files
- Usage:
  - **rm [options] <*file*>...**
- Example:
  - **rm -i *file*** (interactive)
  - **rm -r *directory*** (recursive)
  - **rm -f *file*** (force)

Creating and Removing Directories

- **mkdir** creates directories
- **rmdir** removes empty directories
- **rm -r** recursively removes directory trees

Determining File Content

- Files can contain many types of data
- Check file type with file before opening to determine appropriate command or application to use
- **file [options] <filename>...**

End of Lecture 4

- Questions and Answers
- Summary
  - Files can be managed graphically using **nautilus**
  - Essential command-line file management tools include
    - **cd** to change directories
    - **ls** to list directory contents
    - **cp** to copy files
    - **mv** to move or rename files
    - **rm** to remove files
    - **rm -rf** to remove directories

Lecture 5

Users, Groups and Permissions

Objectives

Upon completion of this unit, you should be able to:

- Explain the Linux security model
- Explain the purpose of user and group accounts
- Read and set file permissions

Users

- Every user is assigned a unique User ID number (*UID*)
  - UID 0 identifies root
  - User accounts normally start at UID 500
- Users' names and UIDs are stored in `/etc/passwd`
- Users are assigned a home directory and a program that is run when they log in (usually a shell)
- Users cannot read, write or execute each others' files without permission

Groups

- Users are assigned to groups
- Each group is assigned a unique Group ID number (*gid*)
- GIDs are stored in `/etc/group`
- Each user is given their own private group
  - Can be added to other groups for additional access
- All users in a group can share files that belong to the group

Linux File Security

- Every file is owned by a UID and a GID
- Every process runs as a UID and one or more GIDs
  - Usually determined by who runs the process
- Three access categories:
  - Processes running with the same UID as the file (*user*)
  - Processes running with the same GID as the file (*group*)
  - All other processes (*other*)

Permission Precedence

- If UID matches, *user* permissions apply
- Otherwise, if GID matches, *group* permissions apply
- If neither match, *other* permissions apply

Viewing Permissions from the Command-Line

- File permissions may be viewed using **ls -l**

```
$ ls -l /bin/login
-rwxr-xr-x 1 root root 19080 Apr 1 18:26 /bin/login
```

- Four symbols are used when displaying permissions:
  - `r`: permission to read a file or list a directory's contents
  - `w`: permission to write to a file or create and remove files from a directory
  - `x`: permission to execute a program or change into a directory and do a long listing of the directory
  - `-`: no permission (in place of the `r`, `w`, or `x`)

Changing File Ownership

- Only root can change a file's owner
- Only root or the owner can change a file's group
- Ownership is changed with **chown**:
  - **chown [-R]** ***user_name file*|*directory*** **...**
- Group-Ownership is changed with **chgrp**:
  - **chgrp [-R]** ***group_name file*|*directory*** **...**

Changing Permissions - Symbolic Method

- To change access modes:
  - **chmod [-*OPTION*]... *mode*[,*mode*] *file*|*directory* ...**
- *mode* includes:
  - **u**,**g** or **o** for user, group and other
  - **+ -** or **=** for grant, deny or set
  - **r**, **w** or **x** for read, write and execute
- Options include:
  - **-R** Recursive
  - **-v** Verbose
  - **--reference** Reference another file for its mode
- Examples:
  - **chmod ugo+r *file***: Grant read access to all for *file*
  - **chmod o-wx *dir***: Deny write and execute to others for *dir*
  - **chmod --reference file1 file2**: Get the mode from `file1` and place it on `file2`

Changing Permissions - Numeric Method

- Uses a three-digit mode number
  - first digit specifies owner's permissions
  - second digit specifies group permissions
  - third digit represents others' permissions
- Permissions are calculated by adding:
  - 4 (for read)
  - 2 (for write)
  - 1 (for execute)
- Example:
  - **chmod 640 myfile**

Changing Permissions - Nautilus

- Nautilus can be used to set the permissions and group membership of files and directories.
    - In a Nautilus window, right-click on a file
    - Select Properties from the context menu
    - Select the Permissions tab

End of Lecture 5

- Questions and Answers
- Summary
  - All files are owned by one user and one group
  - The mode of a file is made up of three permissions: those of the user, the group and all others
  - Three permissions may be granted or denied: read, write and execute

Lecture 6

Using the bash Shell

Objectives

Upon completion of this unit, you should be able to:

- Use command-line shortcuts
- Use command-line expansion
- Use history and editing tricks
- Use the **gnome-terminal**
- Write simple shell scripts
- Set and reference shell variables

Command Line Shortcuts
File Globbing

- *Globbing* is wild card expansion:
  - **`*`** - matches zero or more characters
  - **`?`** - matches any single character
  - **`[0-9]`** - matches a range of numbers
  - **`[abc]`** - matches any one of the characters in the list
  - **`[^abc]`** - matches any one character except those in the list
  - **`[:alpha:]`** - characters in a predefined character class can be matched
- **`glob(7)`**

Command Editing Tricks

- **Ctrl-a** moves to beginning of line
- **Ctrl-e** moves to end of line
- **Ctrl-u** deletes to beginning of line
- **Ctrl-k** deletes to end of line
- **Ctrl-*arrow*** moves left or right by word

Command Line Expansion
The tilde

- Tilde (~)
- May refer to your home directory

```
$ cat ~/.bash_profile
```

- May refer to another user's home directory

```
$ ls ~julie/public_html
```

Command Line Expansion
Commands and Braced Sets

- Command Expansion: **$()** or ``
  - Prints output of one command as an argument to another

```
$ echo "This system's name is $(hostname)"
This system's name is server100.example.com
```

- Brace Expansion: **{ }**
  - Shorthand for printing repetitive strings

```
$ echo file{1,3,5}
file1 file3 file5
$ rm -f file{1,3,5}
```

Bash Variables

- Variables are named values
  - Useful for storing data or command output
- Set with ***VARIABLE*=*VALUE***
- Referenced with **${*VARIABLE*}**

```
$ HI="Hello, and welcome to $(hostname)."
$ echo ${HI}
Hello, and welcome to stationX.
```

6-
5

RH033-RHEL5u4-en-8-20090923/a5424ec6

Environment Variables

- Bash variables are *local* to a single shell by default
    - Set with ***VARIABLE=VALUE***
- *Environment variables* are inherited by child shells
    - Set with **export** ***VARIABLE=VALUE***
    - Accessed by some programs for configuration

Some Common Variables

- Configuration variables
  - `PS1`: Appearance of the **bash** prompt
  - `HISTFILESIZE`: Number of commands in **bash** history
  - `PATH`: Directories to look for executables in
  - `EDITOR`: Default text editor
- Information variables
  - `HOME`: User's home directory
  - `EUID`: User's *effective UID*

Aliases

- Aliases let you create shortcuts to commands

```
$ alias dir='ls -laF'
```

- Use **alias** by itself to see all set aliases
- Use **alias** followed by an alias name to see alias value

```
$ alias dir
alias dir='ls -laF'
```

How bash Expands a Command Line

1. Shell statements are expanded
   - variables, command-substitution, aliases, etc
   - globbing characters, only if they match
2. I/O re-direction is set up
3. Command is executed
   - Command sees the results of expansion, not the shell characters!

Preventing Expansion

- Backslash (\ ) makes the next character literal

```
$ echo Your cost: \$5.00
Your cost: $5.00
```

- Quoting prevents expansion
  - Single quotes (') inhibit all expansion
  - Double quotes (") inhibit all expansion, except:
    - **$** (dollar sign) - variable expansion
    - ` (backquotes) - command substitution
    - \ (backslash) - single character inhibition
    - **!** (exclamation point) - history substitution

Scripting Basics

- Shell scripts are text files containing commands to be executed.
- Shell scripts are useful for:
  - Automating commonly used commands
  - Performing system administration and troubleshooting
  - Creating simple applications
  - Manipulation of text or files

Creating Shell Scripts

- Step 1: Create a text file containing commands
  - First line contains the magic *shebang* sequence: #!
    - **`#!/bin/bash`**
- Comment your scripts!
  - Comments start with a #

Creating Shell Scripts continued

- Step 2: Make the script executable:

```
$ chmod u+x myscript.sh
```

- To execute the new script:
  - Place the script file in a directory in the executable path, such as `~/bin` or `/usr/local/bin` -OR-
  - Specify the absolute or relative path to the script on the command line

Sample Shell Script

```
#!/bin/bash
# This script displays some information about your environment

echo "Greetings. The date and time are $(date)"

echo "Your working directory is: $(pwd)"
```

Login vs non-login shells

- Startup is configured differently for login and non-login shells
- Login shells are:
  - Any shell created at login (includes X login)
  - su -
- Non-login shells are:
  - su
  - graphical terminals
  - executed scripts
  - any other bash instances

Bash startup scripts: profile

- Stored in `/etc/profile` (global) and `~/.bash_profile` (user)
- Run for login shells only
- Used for
  - Setting environment variables
  - Running commands (eg mail-checker script)

Bash startup scripts: bashrc

- Stored in `/etc/bashrc` (global) and `~/.bashrc` (user)
- Run for all bash shells
- Used for
  - Setting local variables
  - Defining aliases

Sourcing files

- Changes to profile and bashrc files need to be *sourced*
- Two methods:
  - `.` ***`scriptname`***
  - **`source`** ***`scriptname`***
- Shell scripts can source other files

Bash Exit Tasks

- Stored in `~/.bash_logout` (user)
- Run when a login shell exits
- Used for
  - Creating automatic backups
  - Cleaning out temporary files

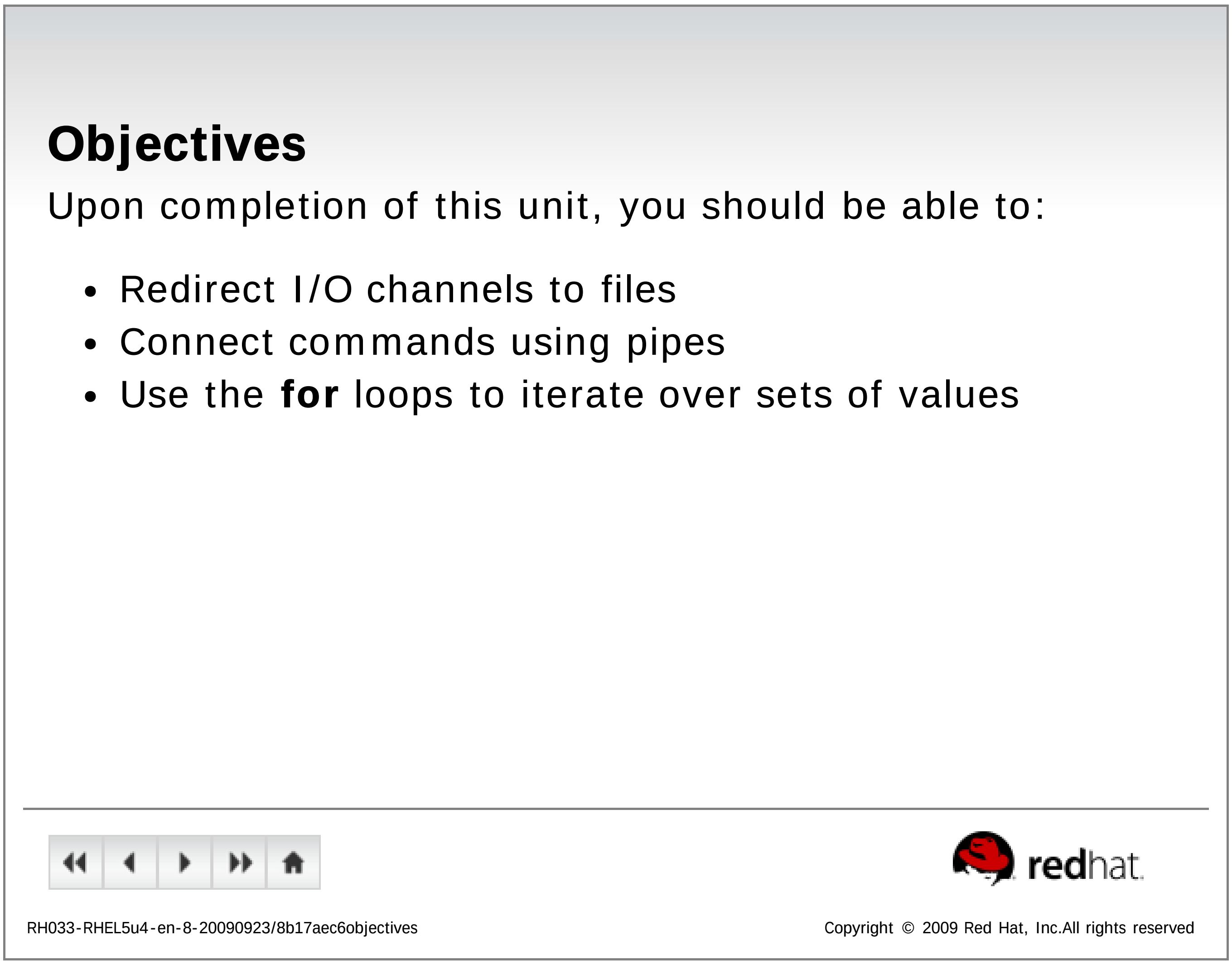
End of Lecture 6

- Questions and Answers
- Summary
  - Command expansion: **`$()`**
  - History recall: **`!`*`string`***, **`!`*`num`***
  - Shell scripting
  - Local variables (`VARNAME=VALUE`) only apply to the shell they are set in
  - Environment variables (`export VARNAME=VALUE`) are inherited by child shells
  - The value of a variable is referenced with `${VARNAME}`

Lecture 7

Standard I/O and Pipes

RH033-RHEL5u4-en-8-20090923/8b17aec6title



Objectives

Upon completion of this unit, you should be able to:

- Redirect I/O channels to files
- Connect commands using pipes
- Use the **for** loops to iterate over sets of values

Standard Input and Output

- Linux provides three I/O channels to Programs
  - Standard input (STDIN) - keyboard by default
  - Standard output (STDOUT) - terminal window by default
  - Standard error (STDERR) - terminal window by default

Redirecting Output to a File

- STDOUT and STDERR can be redirected to files:
  - *`command operator filename`*
- Supported operators include:
  - > Redirect STDOUT to file
  - 2> Redirect STDERR to file
  - &> Redirect all output to file
- File contents are overwritten by default. >> appends.

Redirecting Output to a File
Examples

- This command generates output and errors when run as non-root:

```
$ find /etc -name passwd
```

- Operators can be used to store output and errors:

```
$ find /etc -name passwd > find.out
$ find /etc -name passwd 2> /dev/null
$ find /etc -name passwd > find.out 2> find.err
```

Redirecting STDOUT to a Program (Piping)

- Pipes (the **|** character) can connect commands: ***command1* | *command2***
  - Sends STDOUT of command1 to STDIN of command2 instead of the screen.
  - STDERR is *not* forwarded across pipes
- Used to combine the functionality of multiple tools
  - ***command1* | *command2* | *command3*... etc.**

Useful Pipe Targets

- **less**: View input one page at a time:

  ```
  $ ls -l /etc | less
  ```

  - Input can be searched with **/**

- **mail**: Send input via email:

  ```
  $ echo "test email" | mail -s "test" user@example.com
  ```

- **lpr** : Send input to a printer:

  ```
  $ echo "test print" | lpr
  $ echo "test print" | lpr -P printer_name
  ```

Combining Output and Errors

- Some operators affect both STDOUT and STDERR
  - **&>**: Redirects all output:

    ```
    $ find /etc -name passwd &> find.all
    ```

  - **2>&1**: Redirects STDERR to STDOUT
    - Useful for sending all output through a pipe

    ```
    $ find /etc -name passwd 2>&1 | less
    ```

  - **()**: Combines STDOUTs of multiple programs

    ```
    $ ( cal 2007 ; cal 2008 ) | less
    ```

Redirecting to Multiple Targets (tee)

- `$` ***`command1`*** **`| tee`** ***`filename`*** **`|`** ***`command2`***
- Stores STDOUT of *`command1`* in *`filename`*, then pipes to *`command2`*
- Uses:
  - Troubleshooting complex pipelines
  - Simultaneous viewing and logging of output

Redirecting STDIN from a File

- Redirect standard input with **<**
- Some commands can accept data redirected to STDIN from a file:

```
$ tr 'A-Z' 'a-z' < .bash_profile
```

  - This command will translate the uppercase characters in `.bash_profile` to lowercase
- Equivalent to:

```
$ cat .bash_profile | tr 'A-Z' 'a-z'
```

Sending Multiple Lines to STDIN

- Redirect multiple lines from keyboard to STDIN with **<<*WORD***
  - All text until ***WORD*** is sent to STDIN
  - Sometimes called a *heretext*

```
$ mail -s "Please Call" jane@example.com <<END
> Hi Jane,
>
> Please give me a call when you get in. We may need
> to do some maintenance on the server.
>
> Details when you're on-site,
> Boris
> END
```

Scripting: for loops

- Performs actions on each member of a set of values
- Example:

```
for NAME in joe jane julie
do
        ADDRESS="$NAME@example.com"
        MESSAGE='Projects are due today!'
        echo $MESSAGE | mail -s Reminder $ADDRESS
done
```

Scripting: for loops continued

- Can also use command-output and file lists:
  - **for num in $(seq 1 10)**
    - Assigns 1-10 to $num
    - **seq X Y** prints the numbers X through Y
  - **for file in *.txt**
    - Assigns names of text files to $file

End of Lecture 7

- Questions and Answers
- Summary
    - Standard I/O channels
    - File redirection
        - Standard input (**<**)
        - Standard Output (**>**)
        - Standard Error (**2>**)
    - Pipes redirect standard output to standard input
    - **for** loops can perform commands on items from a program's standard output or an explicit list

Lecture 8

Text Processing Tools

Objectives

Upon completion of this unit, you should be able to:

- Use tools for extracting, analyzing and manipulating text data

Tools for Extracting Text

- File Contents: **less** and **cat**
- File Excerpts: **head** and **tail**
- Extract by Column or Field: **cut**
- Extract by Keyword: **grep**

Viewing File Contents
less and cat

- **cat**: dump one or more files to STDOUT
  - Multiple files are con*cat*enated together
- **less**: view file or STDIN one page at a time
  - Useful commands while viewing:
    - ***/text*** searches for *text*
    - **n**/**N** jumps to the next/previous match
    - **v** opens the file in a text editor (**vi** by default)
  - **less** is the pager used by **man**

Viewing File Excerpts
head and tail

- **head**: Display the first 10 lines of a file
  - Use **-n** to change number of lines displayed
- **tail**: Display the last 10 lines of a file
  - Use **-n** to change number of lines displayed
  - Use **-f** to "follow" subsequent additions to the file
    - Very useful for monitoring log files!

Extracting Text by Keyword
grep

- Prints lines of files or STDIN where a pattern is matched

```
$ grep 'john' /etc/passwd
$ date --help | grep year
```

- Use **-i** to search case-insensitively
- Use **-n** to print line numbers of matches
- Use **-v** to print lines *not* containing pattern
- Use **-A*X*** to include the *X* lines after each match
- Use **-B*X*** to include the *X* lines before each match
- Use **-r** to recursively search a directory
- Use **--color** to highlight the match in color

Extracting Text by Column or Field
cut

- Display specific columns of file or STDIN data

```
$ cut -d: -f1 /etc/passwd
$ grep root /etc/passwd | cut -d: -f7
```

- Use **-d** to specify the column delimiter (default is TAB)
- Use **-f** to specify the column to print
- Use **-c** to cut by characters

```
$ cut -c2-5 /usr/share/dict/words
```

Tools for Analyzing Text

- Text Stats: **wc**
- Sorting Text: **sort**
- Comparing Files: **diff**
- Spell Check: **aspell**

Gathering Text Statistics
wc (word count)

- Counts words, lines, bytes and characters
- Can act upon a file or STDIN

```
$ wc story.txt
39     237    1901 story.txt
```

- Use **-l** for only line count
- Use **-w** for only word count
- Use **-c** for only byte count
- Use **-m** for character count (not displayed)

Sorting Text
sort

- Sorts text to STDOUT - original file unchanged

```
$ sort [options] file(s)
```

- Common options
  - **-r** performs a reverse (descending) sort
  - **-n** performs a numeric sort
  - **-f** ignores (folds) case of characters in strings
  - **-u** (unique) removes duplicate lines in output
  - **-t *c*** uses *c* as a field separator
  - **-k *X*** sorts by *c*-delimited field *X*
    - Can be used multiple times

Eliminating Duplicate Lines
sort and uniq

- **sort -u**: removes duplicate lines from input
- **uniq**: removes duplicate *adjacent* lines from input
  - Use **-c** to count number of occurrences
  - Use with **sort** for best effect:

```
$ sort userlist.txt | uniq -c
```

Comparing Files
diff

- Compares two files for differences

```
$ diff foo.conf-broken foo.conf-works
5c5
<   use_widgets = no
---
>   use_widgets = yes
```

  - Denotes a difference (change) on line 5
- Use **gvimdiff** for graphical **diff**
  - Provided by `vim-X11` package

Spell Checking with aspell

- Interactively spell-check files:

```
$ aspell check letter.txt
```

- Non-interactively list and count mis-spelled words in STDIN

```
$ aspell list < letter.txt
$ aspell list < letter.txt | sort -u | wc -l
```

Tools for Manipulating Text
tr and sed

- Alter (**tr**anslate) Characters: **tr**
  - Converts characters in one set to corresponding characters in another set
  - Only reads data from STDIN

    ```
    $  tr 'a-z' 'A-Z' < lowercase.txt
    ```

- Alter Strings: **sed**
  - **s**tream **ed**itor
  - Performs search/replace operations on a stream of text
  - Normally does not alter source file
  - Use **-i.bak** to back-up and alter source file

sed
Examples

- Quote search and replace instructions!
- **sed** addresses
  - **sed 's/dog/cat/g' pets**
  - **sed '1,50s/dog/cat/g' pets**
  - **sed '/digby/,/duncan/s/dog/cat/g' pets**
- Multiple **sed** instructions
  - **sed -e 's/dog/cat/' -e 's/hi/lo/' pets**
  - **sed -f myedits pets**

Special Characters for Complex Searches
Regular Expressions

- **^** represents beginning of line
- **$** represents end of line
- Character classes as in **bash**:
  - **`[abc]`**, **`[^abc]`**
  - **`[[:upper:]]`**, **`[^[:upper:]]`**
- Used by:
  - **grep**, **sed**, **less**, others

End of Lecture 8

- Questions and Answers
- Summary
  - Extracting Text
    - **cat**, **less**, **head**, **tail**, **grep**, **cut**
  - Analyzing Text
    - **wc**, **sort**, **uniq**, **diff**,
  - Manipulating Text
    - **tr**, **sed**
  - Special Search Characters
    - **^**, **$**, **[abc]**, **[^abc]**, **[[:alpha:]]**, **[^[:alpha:]]**, etc.

Lecture 9

vim: An Advanced Text Editor

RH033-RHEL5u4-en-8-20090923/66ec118dtitle

Objectives

Upon completion of this unit, you should be able to:

- Use the three primary modes of **vi** and **vim**
- Navigate text and enter Insert mode
- Change, delete, yank, and put text
- Undo changes
- Search a document
- Save and exit

Introducing vim

- Newer version of **vi**, the standard Unix text editor
  - Executing **vi** runs **vim** by default
- **gvim**: Graphical version of **vim**
  - Applications + Programming -> Vi IMproved
  - Provided by `vim-X11` package
- Advantages:
  - Speed: Do more with fewer keystrokes
  - Simplicity: No dependence on mouse/GUI
  - Availability: Included with most Unix-like OSes
- Disadvantages
  - Difficulty: Steeper learning curve than simpler editors
    - Key bindings emphasize speed over intuitiveness

vim: A Modal Editor

- Keystroke behavior is dependent upon **vim**'s "mode"
- Three main modes:
  - Command Mode (default): Move cursor, cut/paste text, change mode
  - Insert Mode: Modify text
  - Ex Mode: Save, quit, etc.
- **Esc** exits current mode
- **EscEsc** always returns to command mode

vim Basics

- To use **vim**, you must learn to:
  - Open a file
  - Modify a file (insert mode)
  - Save a file (ex mode)

Opening a file in vim

- To start **vim**:
  - **vim *filename***
  - If the file exists, the file is opened and the contents are displayed
  - If the file does not exist, **vi** creates it when the edits are saved for the first time

Modifying a File
Insert Mode

- **`i`** begins insert mode at the cursor
- Many other options exist
  - **`A`** append to end of line
  - **`I`** insert at beginning of line
  - **`o`** insert new a line (below)
  - **`O`** insert new line (above)

Saving a File and Exiting vim
Ex Mode

- Enter Ex Mode with **:**
  - Creates a command prompt at bottom-left of screen
- Common write/quit commands:
  - **`:w`** writes (saves) the file to disk
  - **`:wq`** writes and quits
  - **`:q!`** quits, even if changes are lost

Using Command Mode

- Default mode of **vim**
- Keys describe movement and text manipulation commands
- Commands repeat when preceded by a number
- Example
  - **Right Arrow** moves right one character
  - **5**, **Right Arrow** moves right five characters

Moving Around
Command Mode

- Move by character: Arrow Keys, **h**, **j**, **k**, **l**
    - Non-arrow keys useful for remote connections to older systems
- Move by word: **w**, **b**
- Move by sentence: **)**, **(**
- Move by paragraph: **}**, **{**
- Jump to line *x*: ***x*G** or **:*x***
- Jump to end: **G**

Search and Replace
Command Mode and EX mode

- Search as in **less**
  - **/**, **n**, **N**
- Search/Replace as in **sed**
  - Affects current line by default
  - Use ***x,y*** ranges or **%** for every line
    - **`:1,5s/cat/dog/`**
    - **`:%s/cat/dog/gi`**

Manipulating Text
Command Mode

	Change (replace)	Delete (cut)	Yank (copy)
Line	`cc`	`dd`	`yy`
Letter	`cl`	`dl`	`yl`
Word	`cw`	`dw`	`yw`
Sentence ahead	`c)`	`d)`	`y)`
Sentence behind	`c(`	`d(`	`y(`
Paragraph above	`c{`	`d{`	`y{`
Paragraph below	`c}`	`d}`	`y}`

Put (paste)

- Use **p** or **P** to put (paste) copied or deleted data
- For line oriented data:
  - **p** puts the data below the current line
  - **P** puts the data above the current line
- For character oriented data:
  - **p** puts the data after the cursor
  - **P** puts the data before the cursor

Undoing Changes
Command Mode

- **u** undo most recent change
- **U** undo all changes to the current line since the cursor landed on the line
- **Ctrl-r** redo last "undone" change

Visual Mode

- Allows selection of blocks of text
  - *v* starts character-oriented highlighting
  - *V* starts line-oriented highlighting
  - Activated with mouse in **gvim**
- Visual keys can be used in conjunction with movement keys:
  - **`w`**, **`)`**, **`}`**, arrows, etc.
- Highlighted text can be deleted, yanked, changed, filtered, search/replaced, etc.

Using multiple "windows"

- Multiple documents can be viewed in a single **vim** screen
  - **Ctrl-w**, **s** splits the screen horizontally
  - **Ctrl-w**, **v** splits the screen vertically
  - **Ctrl-w**, ***Arrow*** moves between windows
- Ex-mode instructions always affect the current window
- **`:help windows`** displays more window commands

Configuring vi and vim

- Configuring on the fly
  - **`:set`** or **`:set all`**
- Configuring permanently
  - `~/.vimrc` or `~/.exrc` (do not include the colon [:] in these files)
- A few common configuration items
  - **`:set number`**
  - **`:set autoindent`**
  - **`:set textwidth=65 (vim only)`**
  - **`:set wrapmargin=15`**
  - **`:set ignorecase`**
- Run **`:help option-list`** for a complete list

Learning more

- **vi**/**vim** built-in help
  - **`:help`**
  - **`:help`** ***`topic`***
  - Use **`:q`** to exit help
- **vimtutor** command

End of Lecture 9

- Questions and Answers
- Summary
  - Use the three primary modes of **vi** and **vim**
  - Move the cursor and enter Insert mode
  - Change, delete, yank, and put text
  - Undo changes
  - Search a document
  - Save and exit

Lecture 10

Investigating and Managing Processes

Objectives

Upon completion of this unit, you should be able to:

- Explain what a process is
- Describe how to manage processes
- Use job control tools
- Schedule recurring jobs
- Employ decision making constructs in shell scripts

What is a Process?

- A process is a set of instructions loaded into memory
  - Numeric *Process ID* (PID) used for identification
  - UID, GID and SELinux context determines filesystem access
    - Normally inherited from the executing user

Listing Processes

- View Process information with **ps**
  - Shows processes by the current user on the current terminal by default
  - **-e** shows all processes
  - **-u *user*** shows all processes by ***user***
  - **-F** prints extra information
  - **-H** indents child processes
  - **-o PROPERTY1,PROPERTY2,...** prints custom information:
    - **pid**, **comm**, **%cpu**, **%mem**, **state**, **tty**, **euser**, **ruser**, etc.
- Example:
  - **`ps -eo pid,%cpu,comm`**

Finding Processes

- Most flexible: **ps** ***options*** **|** ***other commands***
  ps -eo comm,tty | grep ttyS0
- By predefined patterns: **pgrep**

```
$ pgrep -U root
```

```
$ pgrep -G student
```

- By exact program name: **/sbin/pidof**

```
$ ps -p $(/sbin/pidof bash)
```

Signals

- Sent directly to processes, no user-interface required
- Programs associate actions with each signal
- Signals are specified by name or number when sent:
  - Signal 15, TERM (default) - Terminate cleanly
  - Signal 9, KILL - Terminate immediately
  - Signal 1, HUP - Re-read configuration files
  - **man 7 signal** shows complete list

Sending Signals to Processes

- By PID: **kill [-*signal*] *pid* ...**
- By Name: **killall [-*signal*] *comm* ...**
- By pattern: **pkill [-*signal*] *pattern***

Scheduling Priority

- Scheduling priority determines access to the CPU
- Priority is affected by a process' *nice value*
- Values range from -20 to 19 but default to 0
  - Lower nice value means higher CPU priority
- Viewed with **ps -o comm,nice**

Altering Scheduling Priority

- Nice values may be altered...
  - When starting a process:

    ```
    $ nice -n 5 command
    ```

  - After starting:

    ```
    $ renice 5 PID
    ```

- Only root may decrease nice values

Interactive Process Management Tools

- CLI: **top**
- GUI: **gnome-system-monitor**
- Capabilities
  - Display real-time process information
  - Allow sorting, killing and re-nicing

Job Control

- Run a process in the background
  - Append an ampersand to the command line: **firefox &**
- Temporarily halt a running program
  - Use **Ctrl**-**z** or send signal 19 (STOP)
- Manage background or suspended jobs
  - List job numbers and names: **jobs**
  - Resume in the background: **bg [%*jobnum*]**
  - Resume in the foreground: **fg [%*jobnum*]**
  - Send a signal: **kill [-*SIGNAL*] [%*jobnum*]**

Exit Status

- Processes report success or failure with an exit status
  - 0 for success, 1-255 for failure
  - **$?** stores the exit status of the most recent command
  - **exit [*num*]** terminates and sets status to *num*
- Example:

```
$ ping -c1 -W1 station999 &> /dev/null
$ echo $?
2
```

Conditional Execution Operators

- Commands can be run conditionally based on exit status
  - **&&** represents conditional AND THEN
  - **||** represents conditional OR ELSE
- Examples:

```
$ grep -q no_such_user /etc/passwd || echo 'No such user'
No such user

$ ping -c1 -W2 station1 &> /dev/null  \
>        && echo "station1 is up"         \
>        || { echo 'station1 is unreachable'; exit 1; }
station1 is up
```

The test Command

- Evaluates boolean statements for use in conditional execution
  - Returns 0 for true
  - Returns 1 for false
- Examples in long form:

```
$ test "$A" =  "$B" && echo "Strings are equal"
$ test "$A" -eq "$B" && echo "Integers are equal"
```

- Examples in shorthand notation:

```
$ [ "$A" =  "$B" ] && echo "Strings are equal"
$ [ "$A" -eq "$B" ] && echo "Integers are equal"
```

File Tests

- File tests:
  - **-f** tests to see if a file exists and is a regular file
  - **-d** tests to see if a file exists and is a directory
  - **-x** tests to see if a file exists and is executable

```
[ -f ~/lib/functions ] && source
~/lib/functions
```

Scripting: if Statements

- Execute instructions based on the exit status of a command

```
if ping -c1 -w2 station1 &> /dev/null; then
    echo 'Station1 is UP'
elif grep "station1" ~/maintenance.txt &> /dev/null; then
    echo 'Station1 is undergoing maintenance'
else
    echo 'Station1 is unexpectedly DOWN!'
        exit 1
fi
```

End of Lecture 10

- Questions and Answers
- Summary
    - A process is any set of instructions in memory
    - Processes are managed with: **ps**, **kill**, **top**, **gnome-system-monitor**
    - Suspend jobs with **Ctrl**-**z**, manage with **fg**, **bg**
    - Every process returns a numeric *exit status* upon exit
    - **test** returns 0 or 1 depending on parameters
    - **if**/**else**, **&&** and **||** can execute commands based on predecessors' exit status

Lecture 11

Basic System Configuration Tools

RH033-RHEL5u4-en-8-20090923/4175ab48title

Objectives

Upon completion of this unit, you should be able to:

- Configure the network
- Configure and send text to a printer
- Set the system's date and time
- Schedule time-delayed tasks
- Schedule recurring tasks
- Know how to handle input with the **read** command and positional parameters

TCP/IP Network Configuration

- Important network settings:
  - IP Configuration
  - Device Activation
  - DNS Configuration
  - Default Gateway

Managing Ethernet Connections

- Network interfaces are named sequentially: `eth0`, `eth1`, etc.
  - Multiple addresses can be assigned to a device with *aliases*
  - Aliases are labeled `eth0:1`, `eth0:2`, etc.
  - Aliases are treated like separate interfaces
- View interface configuration with **/sbin/ip addr show [eth*X*]**
- Enable interface with **/sbin/ifup eth*X***
- Disable interface with **/sbin/ifdown eth*X***

Graphical Network Configuration
system-config-network

- System->Administration->Network
  - Activate/Deactivate interfaces
  - Assign IP Addresses/DHCP
  - Modify DNS settings
  - Modify gateway address

Network Configuration Files
Ethernet Devices

- Device configuration is stored in text files
  - `/etc/sysconfig/network-scripts/ifcfg-eth`*`X`*
  - Complete list of options in `/usr/share/doc/initscripts-*/sysconfig.txt`

Dynamic Configuration	Static Configuration

DEVICE=eth*X*
HWADDR=0:02:8A:A6:30:45
BOOTPROTO=dhcp
ONBOOT=yes
Type=Ethernet
	DEVICE=eth*X*
	HWADDR=0:02:8A:A6:30:45
	IPADDR=192.168.0.123
	NETMASK=255.255.255.0
	GATEWAY=192.168.0.254
	ONBOOT=yes
	Type=Ethernet

Network Configuration Files

Other Global Network Settings

- Global Settings in `/etc/sysconfig/network`
  - Many may be provided by DHCP
  - GATEWAY can be overridden in ifcfg file

```
NETWORKING=yes
HOSTNAME=server100.example.com
GATEWAY=192.168.0.254
```

Network Configuration Files

DNS Configuration

- Domain Name Service translates hostnames to network addresses
- Server address is specified by dhcp or in `/etc/resolv.conf`

```
search example.com remote.test
nameserver 192.168.0.254
nameserver 192.168.1.254
```

Printing in Linux

- Printers may be local or networked
- Print requests are sent to queues
- Queued jobs are sent to the printer on a first come first served basis
- Jobs may be canceled before or during printing

system-config-printer

- System->Administration->Printing
- Supported printer connections:
  - Local (parallel, serial or usb)
  - Unix/Linux print server
  - Windows print server
  - Netware print server
  - HP JetDirect
- Configuration stored in `/etc/cups/printers.conf`

Printing Commands

- **lpr** sends a job to the queue to be printed
  - Accepts ASCII, PostScript, PDF, others
- **lpq** views the contents of the queue
- **lprm** removes a job from the queue
- System V printing commands such as **lp**, **lpstat** and **cancel** are also supported

Printing Utilities

- **evince** views PDF and PostScript documents
- **lpstat -a** lists configured printers
- **enscript** and **a2ps** convert text to PostScript
- **ps2pdf** converts PostScript to PDF
- **mpage** prints multiple pages per sheet

Setting the System's Date and Time

- GUI: **system-config-date**
  - System->Administration->Date & Time
  - Can set date/time manually or use NTP
  - Additional NTP servers can be added
  - Can use local time or UTC
- CLI: **date [MMDDhhmm[[CC]YY][.ss]]**

  -
    ```
    # date 01011330
    # date 010113302007.05
    ```

Scheduling Commands To Execute Later

- One-time jobs use **at**, recurring jobs use **crontab**

Create	**at** ***time***	**crontab -e**
List	**at -l**	**crontab -l**
Details	**at -c** ***jobnum***	N/A
Remove	**at -d** ***jobnum***	**crontab -r**
Edit	N/A	**crontab -e**

- Non-redirected output is mailed to the user
- *root* can modify jobs for other users

Crontab File Format

- Entry consists of five space-delimited fields followed by a command line
  - One entry per line, no limit to line length
- Fields are minute, hour, day of month, month, and day of week
- Comment lines begin with #
- See **man 5 crontab** for details

Scripting: Taking input with positional Parameters

- Special variables that hold the command-line arguments to the script
  - Position-related names: $1, $2, $3, etc.
  - Arguments are space-delimited
  - Words can be grouped into a single argument with quotes
- Normally assigned to more meaningful variable names to improve clarity
- $* holds all command-line arguments
- $# holds the number of command-line arguments

Scripting: Taking input with the read command

- Use **`read`** to assign input values to one or more shell variables:
  - **`-p`** designates prompt to display
  - **`read`** reads from standard input and assigns one word to each variable
  - Any leftover words are assigned to the last variable
  - **`read -p "Enter a filename: " FILE`**

End of Lecture 11

- Questions and Answers
- Summary
    - **system-config-network** configures /etc/sysconfig/network-scripts/*
    - **ifup**, **ifdown**
    - **lpr** sends text to the printer
    - **date** configures date/time from CLI
    - **system-config-date** configures date/time from GUI
    - Use **at** to schedule time-delayed tasks
    - Use **crontab -e** to schedule recurring tasks
    - Administrative tasks may be defined in /etc/cron.d/cron.*
    - **read *VAR*** sets variable from STDIN
    - **$1**, **$2**, etc. map to command-line arguments
    - **$#** represents the number of arguments to a script

Lecture 12

Finding and Processing Files

Objectives

Upon completion of this unit, you should be able to:

- Use **locate**
- Use **find**
- Use **the Gnome Search tool**

The Gnome Search Tool

- Places->Search for Files...
- Graphical tool for searching by
  - name
  - content
  - owner/group
  - size
  - modification time

locate

- Queries a pre-built database of paths to files on the system
  - Database must be updated by administrator
  - Full path is searched, not just filename
- May only search directories where the user has read and execute permission

locate Examples

- **`locate passwd`**
  - Search for files with "passwd" in the name or path
- Useful options
  - **`-i`** performs a case-insensitive search
  - **`-n`** ***X*** lists only the first *X* matches

find

- **find [*dir1* ...] [*criteria*...] [*action*...]**
- Searches directory trees in real-time
  - Slower but more accurate than **locate**
  - CWD is used if no starting directory given
  - All files are matched if no criteria given
- Can execute commands on found files
- Can apply boolean logic to criteria
- May only search directories where the user has read and execute permission

Basic find Examples

- **`find -name snow.png`**
  - Search for files named `snow.png` in the current directory
- **`find -iname snow.png`**
  - Case-insensitive search for files named `snow.png`, `Snow.png`, `SNOW.PNG`, etc. in the current directory
- **`find / -name '*.txt'`**
  - Search for files anywhere on the system that end in `.txt`
  - Wild cards should always be quoted to avoid unexpected results
- **`find /etc -name '*pass*'`**
  - Search for files in `/etc/` that contain pass in their name
- **`find /home -user joe -group joe`**
  - Search for files owned by the user *joe* and the group *joe* in `/home/`

find and Logical Operators

- Criteria are ANDed together by default.
- Can be OR'd or negated with **-o** or **-not**
- Parentheses can be used to determine logic order, but must be escaped in bash
  - **`find -user joe -not -group joe`**
  - **`find -user joe -o -user jane`**
  - **`find -not \( -user joe -o -user jane \)`**

find and Permissions

- Can match ownership by name or id
  - **find / -user joe -o -uid 500**
- Can match octal or symbolic permissions
  - **find -perm 755**
    - matches if mode is *exactly* 755
  - **find -perm +222**
    - matches if *anyone* can write
  - **find -perm -222**
    - matches if *everyone* can write
  - **find -perm -002**
    - matches if other can write

find and Numeric Criteria

- Many **find** criteria take numeric values
- **`find -size 10M`**
  - Files with a size of *exactly* 10 megabytes
- **`find -size +10M`**
  - Files with a size *over* 10 megabytes
- **`find -size -10M`**
  - Files with a size *less than* 10 megabytes
- Other modifiers are available such as k for KB, G for GB, etc.

find and Access Times

- **find** can match by inode timestamps
  - **-atime** when file was last read
  - **-mtime** when file data last changed
  - **-ctime** when file data or metadata last changed
- Value given is in days
  - `find /tmp -ctime +10`
    - Files changed more than 10 days ago
- Can use a value of minutes
  - **-amin**
  - **-mmin**
  - **-cmin**
  - `find /etc -amin -60`

Executing Commands with find

- Commands can be executed on found files
  - Command must be preceded with **`-exec`** or **`-ok`**
    - **`-ok`** prompts before acting on each file
  - Command must end with **`Space\;`**
  - Can use {} as a filename placeholder
  - **`find -size +100M -ok mv {} /tmp/largefiles/ \;`**

find Execution Examples

- Back up configuration files, adding a .orig extension

```
$ find -name '*.conf' -exec cp {} {}.orig \;
```

- Prompt to remove Joe's tmp files that are over 3 days old

```
$ find /tmp -ctime +3 -user joe -ok rm {} \;
```

- Fix other-writable files in your home directory

```
$ find ~ -perm -002 -exec chmod o-w {} \;
```

- Do an **ls -l** style listing of all directories in /home/

```
$ find /home -type d -ls
```

- Find files that end in .sh but are not executable by anyone. For each file, ask to make it executable by everyone

```
$ find -not -perm +111 -name '*.sh' -ok chmod 755 {} \;
```

End of Lecture 12

- Questions and Answers
- Summary
  - Use **locate** to quickly find files that are not new
  - Use **find** to search based on very specific criteria and optionally run commands on matching files
  - Use the **Gnome Search Tool** for an intuitive, but powerful GUI search tool.

RH033-RHEL5u4-en-8-20090923/b6ee4e89summary

Lecture 13

Network Clients

RH033-RHEL5u4-en-8-20090923/68130981title

Objectives

Upon completion of this unit, you should be able to:

- Browse the web
- Exchange email and instant messages
- Access a Linux system remotely
- Transfer files between systems
- Use network diagnostic tools

Web Clients

- GUI and Non-GUI web browsers
- **wget**

Firefox

- Fast, lightweight, feature-rich web browser
  - Tabbed browsing
  - Popup blocking
  - Cookie management
  - Multi-engine search bar
  - Support for many popular plug-ins
  - Themes and Extensions

links

- **links** a non-GUI web browser
  - Provided by the elinks rpm
  - Full support for frames and SSL
  - Examples
    - **links http://www.redhat.com**
    - **links -dump http://www.redhat.com**
    - **links -source http://www.redhat.com**
  - Particularly useful for
    - Connectivity testing when **ping** is blocked
    - File retrieval when you don't remember the full URL to type for **curl** or **wget**

wget

- Retrieves files via HTTP and FTP
- Non-interactive - useful in shell scripts
- Can follow links and traverse directory trees on the remote server - useful for mirroring web and FTP sites

Email and Messaging

- **Evolution**
- Thunderbird
- Mutt
- Pidgin

Graphical Mail Clients

- Available in Red Hat Enterprise Linux Client variant only
- Evolution
  - Flexible email and groupware tool
- Thunderbird
  - Standalone Mozilla email client

Non-GUI Mail Clients

- **mutt**
  - Supports pop, imap and local mailboxes
  - Highly configurable
  - Mappable hot keys
  - Message threading and colorizing
  - GnuPG integration
  - Context-sensitive help with '?'

Pidgin: Instant Messaging

- Formerly known as **GAIM**
- Available in Red Hat Enterprise Linux Client variant only
- Multi-protocol Instant messaging client
- Supports AIM, MSN, ICQ, Yahoo, Jabber, Gadu-Gadu, SILC, GroupWise Messenger, IRC and Zephyr networks
- Plugins can be used to add functionality

Remote Access and File Transfer with Nautilus

- Places->Connect to Server
- Graphically browse with multiple protocols
- Allows drag-and-drop file transfers
- Supported connection types: FTP, SFTP, SMB, WebDAV, Secure WebDAV
- Can also connect via url:
  - File->Open Location

OpenSSH: Secure Remote Shell

- Secure replacement for older remote-access tools
- Allows authenticated, encrypted access to remote systems
  - **ssh [*user@*]*hostname***
  - **ssh [*user@*]*hostname command***
    - Include **-X** for graphical applications
      - Beware: hostile systems can take advantage of this!
      - Only use **-X** on trusted systems!

scp: Secure File Transfer

- Secure replacement for rcp
- Layered on top of ssh
  - **scp** ***source destination***
  - Remote files can be specified using:
    - **[user@]host:/path/to/file**
  - Use **-r** to enable recursion
  - Use **-p** to preserve times and permissions
  - Use **-C** to compress data stream

rsync: Efficient File Sync

- Efficiently copies files to or from remote systems
- Uses secure **ssh** connections for transport
    - **rsync *.conf barney:/home/joe/configs/**
- Faster than **scp** - copies differences in like files

OpenSSH Key-based Authentication

- Optional, password-less, but still secure, authentication
- Uses two keys generated by **`ssh-keygen`**:
  - *private key* stays on your system
    - Usually passphrase-protected (*recommended*)
  - *public key* is copied to destination with **`ssh-copy-id`**
    - **`ssh-copy-id -i ~/.ssh/id_rsa.pub [user@]host`**

OpenSSH Key-based Authentication continued

- An *authentication agent* stores decrypted private keys
  - Thus, passphrase only needs to be entered once
  - An agent is provided automatically in GNOME
  - Otherwise, run **ssh-agent bash**
- Keys are added to the agent with **ssh-add**

FTP Clients

- CLI: **lftp**

```
$ lftp ftp.example.com
$ lftp -u joe ftp.example.com
```

  - Automated transfers with **lftpget**
- GUI: **gFTP**
  - Applications->Internet->gFTP
  - Allows Drag-and-Drop transfers
  - Anonymous or authenticated access
  - Optional secure transfer via ssh (sftp)

smbclient

- FTP-like client to access SMB/CIFS resources
- Examples:
  - **smbclient -L server100** lists shares on server100
  - **smbclient -U student //server100/homes** accesses a share

Network Diagnostic Tools

- **ping**
- **traceroute**
- **host**
- **dig**
- **netstat**
- **gnome-nettool** (GUI)

End of Lecture 13

- Questions and Answers
- Summary
  - Firefox, Evolution and Mutt
  - Basic network diagnostic tools
  - The importance of secure network clients

RH033-RHEL5u4-en-8-20090923/68130981summary

Lecture 14

Advanced Topics in Users, Groups and Permissions

Objectives

Upon completion of this unit, you should be able to:

- Describe where Linux stores user, group and password information
- Set default permissions
- Use special permissions

User and Group ID Numbers

- User names map to user ID numbers
- Group names map to group ID numbers
- Data stored on the hard disk is stored numerically

`/etc/passwd`, `/etc/shadow`, and `/etc/group` files

- Authentication information is stored in plain text files:
    - `/etc/passwd`
    - `/etc/shadow`
    - `/etc/group`
    - `/etc/gshadow`

User Management Tools

- GUI
  - **system-config-users**
- CLI
  - **useradd**
  - **usermod**
  - **userdel [-r]**

System Users and Groups

- Server programs such as web or print servers typically run as unprivileged users, not as root
  - Examples: `daemon`, `mail`, `lp`, `nobody`
- Running programs in this way limits the amount of damage any single program can do to the system

Monitoring Logins

- Connected users: **w**
- Login and reboot history: **last**
- Failed login attempts: **lastb**
- Most recent logins: **lastlog**

Default Permissions

- Default permission for directories is 777 minus *umask*
- Default permission for files is the directory default without execute permission.
- *umask* is set with the **umask** command
- Non-privileged users' **umask** is 002
  - Files will have permissions of 664
  - Directories will have permissions of 775
- `root`'s **umask** is 022

Special Permissions for Executables

- Special permissions for executables:
  - **suid**: command run with permissions of the *owner* of the command, not executor of the command
  - **sgid**: command runs with group affiliation of the group of the command

Special Permissions for Directories

- Special permissions for directories:
    - sticky bit: files in directories with the sticky bit set can only be removed by the owner and root, regardless of the write permissions of the directory
    - sgid: files created in directories with the sgid bit set have group affiliations of the group of the directory

End of Lecture 14

- Questions and Answers
- Summary
  - User information is stored in `/etc/passwd`
  - Group information is stored in `/etc/group`
  - Special Permissions: Sticky Bit, SetUID, SetGID

RH033-RHEL5u4-en-8-20090923/a38cbe18summary

Lecture 15

The Linux Filesystem In-Depth

Objectives

Upon completion of this unit, you should be able to:

- Describe how filesystem information is organized
- Describe the function of dentries and inodes
- Describe how **cp**, **mv**, and **rm** work at the inode level
- Create symbolic links and hard links
- Access removable media
- Create archives using **tar** and **gzip**

RH033-RHEL5u4-en-8-20090923/14067d4bobjectives

Partitions and Filesystems

- Disk drives are divided into *partitions*
- Partitions are formatted with *filesystems*, allowing users to store data
  - Default filesystem: ext3, the Third Extended Linux Filesystem
  - Other common filesystems:
    - ext2 and msdos (typically used for floppies)
    - iso9660 (typically used for CDs)
    - GFS and GFS2 (typically for SANs)

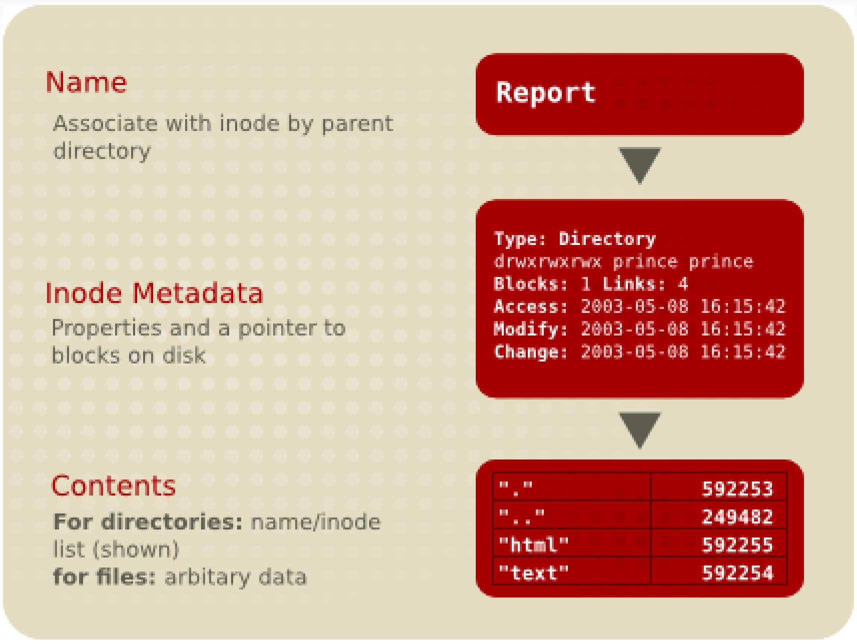
Inodes

- An *inode table* contains a list of all files in an ext2 or ext3 filesystem
- An *inode* (index node) is an entry in the table, containing information about a file (the *metadata*), including:
  - file type, permissions, UID, GID
  - the link count (count of path names pointing to this file)
  - the file's size and various time stamps
  - pointers to the file's data blocks on disk
  - other data about the file

Directories

- The computer's reference for a file is the *inode number*
- The human way to reference a file is by *file name*
- A *directory* is a mapping between the human name for the file and the computer's inode number

Inodes and Directories


Name
Associate with inode by parent directory
Inode Metadata
Properties and a pointer to blocks on disk
Contents
For directories: name/inode list (shown)
for files: arbitary data
Report

```
Type: Directory
drwxrwxrwx prince prince
Blocks: 1 Links: 4
Access: 2003-05-08 16:15:42
Modify: 2003-05-08 16:15:42
Change: 2003-05-08 16:15:42
```

"."	592253
".."	249482
"html"	592255
"text"	592254

cp and inodes

- The **cp** command:
  1. Allocates a free inode number, placing a new entry in the inode table
  2. Creates a dentry in the directory, associating a name with the inode number
  3. Copies data into the new file

mv and inodes

- If the destination of the **mv** command is on the same file system as the source, the **mv** command:
  1. Creates a new directory entry with the new file name
  2. Deletes the old directory entry with the old file name
- Has no impact on the inode table (except for a time stamp) or the location of data on the disk: no data is moved!
- If the destination is a different filesystem, **mv** acts as a copy and remove

rm and inodes

- The **rm** command:
  1. Decrements the link count, thus freeing the inode number to be reused
  2. Places data blocks on the free list
  3. Removes the directory entry
- Data is not actually removed, but will be overwritten when the data blocks are used by another file

Hard Links

- A hard link adds an additional dentry to reference a single file
  - One physical file on the filesystem
  - Each directory references the same inode number
  - Increments the link count
    - The **rm** command decrements the link count
    - File exists as long as at least one link remains
    - When the link count is zero, the file is removed
  - Cannot span drives or partitions
- Syntax:
  - **ln *filename* [*linkname*]**

Symbolic (or Soft) Links

- A symbolic link points to another file
  - **ls -l** displays the link name and the referenced file
    `lrwxrwxrwx 1 joe joe 11 Sep 25 18:02 pf -> /etc/passwd`
  - File type: **l** for symbolic link
  - The content of a symbolic link is the name of the file that it references
- Syntax:
  - **`ln -s filename`** ***`linkname`***

The Seven Fundamental File types

ls -l symbol	File Type
-	regular file
d	directory
l	symbolic link
b	block special file
c	character special file
p	named pipe
s	socket

Checking Free Space

- **baobab** produces graphical usage report by directory
  - Applications->System Tools->Disk Usage Analyzer
- **du** produces text usage report (in kilobytes) by directory
  - Lists size of every file in all sub-directories by default
    - **-h** and **-H** display sizes in easier-to-read units
    - **-s** summarizes sub-directories instead
- **df** produces text usage report (in kilobytes) by filesystem
  - Also takes **-h** and **-H** options
  - **-T** includes filesystem types

Removable Media

- *Mounting* integrates a foreign filesystem into the main tree
- Before accessing, media must be mounted
- Before removing, media must be unmounted
- In Gnome and KDE, devices auto-mount under `/media/`
- In console, root can manually mount devices under `/mnt/`

```
# mkdir /mnt/floppy
# mount /dev/fd0 /mnt/floppy
# umount /dev/fd0
```

- In console, non-root users can use **gnome-mount** and **gnome-umount**

```
$ gnome-mount -t -d /dev/cdrom
$ gnome-mount -t -d /dev/sda
$ gnome-umount -t -d /dev/sdb1
```

CDs and DVDs

- Automatically mounted in Gnome/KDE
  - Accessible from:
    - Computer desktop icon, CD-ROM
    - CD-ROM Desktop icon
    - `/media/`*`disk_label`* or `/media/CDROM`
- Ejected with:
  - Right Click->Eject
  - **eject /dev/cdrom**
- From command-line, use **gnome-mount** and **gnome-umount**

  -
    ```
    $ gnome-mount -t -d /dev/cdrom
    $ gnome-umount -t -d /dev/cdrom
    ```

USB Media

- Detected by the kernel as SCSI devices
  - /dev/sda, /dev/sda*X*, /dev/sdb, /dev/sdb*X*, etc.
- Automatically mounted in Gnome/KDE
  - Similar location as CDs
    - /media/*disk_label* or /media/disk
  - Unmounted with:
    - Right Click->Unmount Volume
    - **umount /dev/sda*X***
- From command-line, use **gnome-mount** and **gnome-umount**

  -
    ```
    $ gnome-mount -t -d /dev/sda1
    $ gnome-umount -t -d /dev/sda1
    ```

Archiving Files and Compressing Archives

- Archiving places many files into one target file
  - Easier to back up, store, and transfer
  - **tar** - standard Linux archiving command
- Archives are commonly compressed
  - Algorithm applied that compresses file
  - Uncompressing restores the original file
  - **tar** natively supports compression using **gzip** and **gunzip**, or **bzip2** and **bunzip2**

Essential tar Options

- Actions (one is required):
  - **-c** create an archive
  - **-t** list an archive
  - **-x** extract files from an archive
- Typically required:
  - **-f** ***archivename*** name of file archive
- Optional:
  - **-z** use **gzip** compression
  - **-j** use **bzip2** compression
  - **-v** be verbose
  - **--xattrs** store SELinux and ACL properties

Creating File Archives: Other Tools

- **zip** and **unzip**
  - Supports **pkzip**-compatible archives
  - Example:

```
zip -r etc.zip /etc
unzip etc.zip
```

- **file-roller**
  - Graphical, multi-format archiving tool

End of Lecture 15

- Questions and Answers
- Summary
  - Linux filesystem structure
  - Using removable media
  - Using unformatted floppies
  - Archiving and compression

Lecture 16

Essential System Administration Tools

Objectives

Upon completion of this unit, you should be able to:

- Explain the process of installing Red Hat Enterprise Linux
- Identify services, their status and be able to manage the runlevels which start and stop them
- Install software using multiple installation methods
- Understand the basic principles of Red Hat Enterprise Linux security, firewalls, and SELinux

Planning an Installation

- What hardware does the system use?
  - Check hardware compatibility
- Read the `RELEASE-NOTES` file on the first DVD/CD or at http://www.redhat.com
  - Provides valuable summary of features and gotchas

Performing an Installation

- Installer can be started from:
  - CD-ROM or DVD-ROM
  - USB Device
  - Network (PXE)
- Supported installation sources:
  - Network Server (ftp, http or nfs)
  - CD-ROM or DVD-ROM
  - Hard Disk

Accessing the Installer

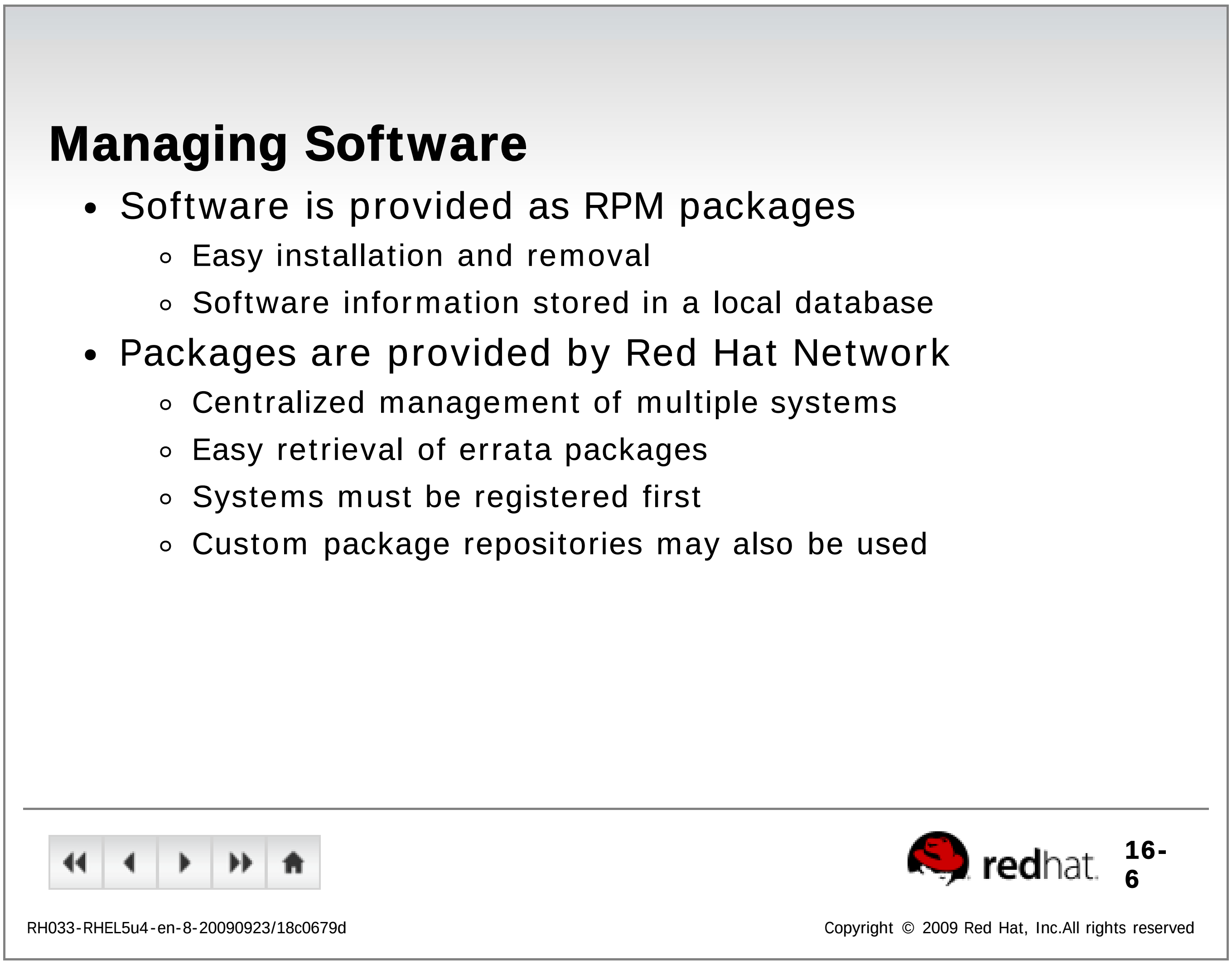
- Graphical installation
  - Default installation type
  - Useful switches: **lowres**, **resolution**, **skipddc**
- VNC based installation
  - Activate with **vnc** and protect the session with **vncpassword=*password***
  - Set network parameters with **ip=*IPAddress*** and **netmask=*NetworkMask***
- Text based installation
  - Started with the **text** switch
  - Menu-based terminal interface
- Serial installation
  - Used automatically when no graphic card is detected
  - Enable with: **serial=*device***

First Boot: Post-Install Configuration

- Configure X Window System if necessary
- Firewall and SELinux Setup
- Kdump setup
- Set date and time
- Register with Red Hat Network and get updated RPMs
- Create a first user
- Configure sound card
- Install additional RPMs or Red Hat documentation from CDROM

Managing Services

- What is a service?
- Graphical Interface to Service Management
  - **system-config-services**
- Command Line Interface to Service Management
  - ***/sbin/service***
  - ***/sbin/chkconfig***



Managing Software

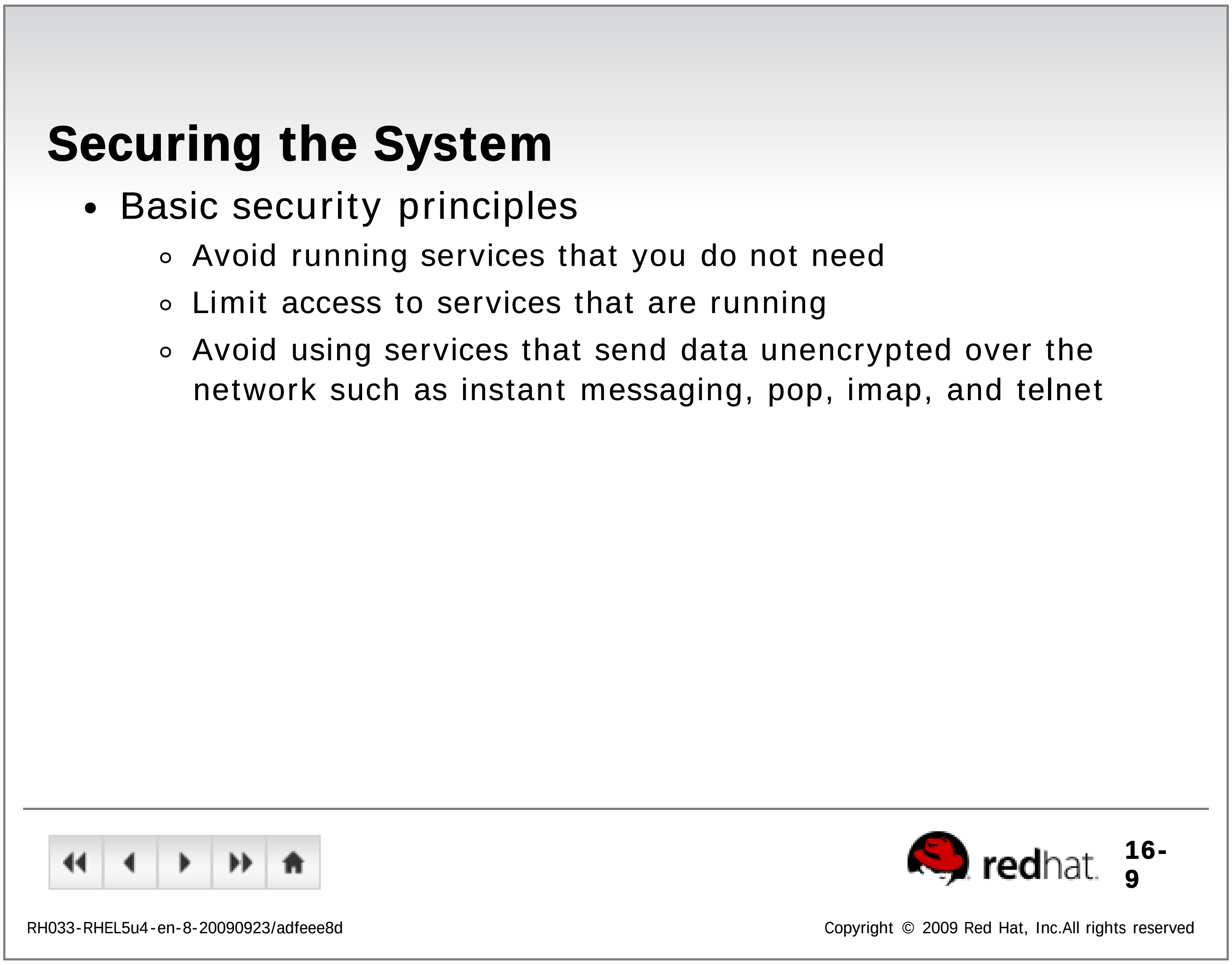
- Software is provided as RPM packages
  - Easy installation and removal
  - Software information stored in a local database
- Packages are provided by Red Hat Network
  - Centralized management of multiple systems
  - Easy retrieval of errata packages
  - Systems must be registered first
  - Custom package repositories may also be used

Graphical Package Management

- **pup**
  - Applications->System Tools->Software Updater
  - List and install software updates
- **pirut**
  - Applications->Add/Remove Software
  - View, install and un-install other packages

The Yum Package Management Tool

- Front-end to **rpm**, replacing **up2date**
- Configuration in `/etc/yum.conf` and `/etc/yum.repos.d/`
- Used to install, remove and list software
  - **yum install** ***packagename***
  - **yum remove** ***packagename***
  - **yum update** ***packagename***
  - **yum info** ***packagename***
  - **yum list available**
  - **yum list installed**



Securing the System

- Basic security principles
  - Avoid running services that you do not need
  - Limit access to services that are running
  - Avoid using services that send data unencrypted over the network such as instant messaging, pop, imap, and telnet

SELinux

- Kernel-level security system
- All processes and files have a *context*
- SELinux *Policy* dictates how processes and files may interact based on context
  - Policy rules cannot be overridden
  - Default policy does not apply to all services

Managing SELinux

- SELinux violations are logged in the System Log
- SELinux can be disabled in an emergency
- Disabling SELinux is discouraged!
- **system-config-selinux**
    - System->Administration->SELinux Management

Packet Filtering

- Network traffic is divided into packets
- Each packet has source/destination data
- Firewalls selectively block packets

Firewall and SELinux Configuration
system-config-securitylevel

- System-> Administration->Security Level and Firewall
  - Selectively allow incoming connections by port
  - Responses to outbound queries always accepted
  - Alternate interface for basic SELinux configuration
- More advanced configuration possible with other tools

End of Lecture 16

- Questions and Answers
- Summary
  - System Installation Process
  - Managing Services
  - Software Installation Tools
  - System Security

Lecture 17

So, What Now?

Objectives

Upon completion of this unit, you should be able to:

- Explore further Red Hat training
- Participate in the Linux community

Next Up...

- RH131: Red Hat Enterprise Linux System Administration
  - Install virtual and physical systems
  - Manage local and centralized user accounts
  - Manage partitions, RAID arrays and logical volumes
  - Advanced package management
  - Troubleshooting
  - More
- Bundled with RHCT exam as RH133

Other Red Hat System Administration Courses

- RH253: Red Hat Linux Networking & Security Administration
- RH300: Red Hat Certified Engineer Rapid Track Course
- Advanced certifications: RHCA, RHCSS, and RHCDS

Red Hat Developer Classes

- RHD251: Red Hat Enterprise Linux Development
- RHD361: Red Hat Enterprise Linux Kernel Internals
- RHD362: Red Hat Enterprise Linux Device Drivers

JBoss Middleware Courses

- JB336: JBoss for Application Administrators
- JB295: JBoss Enterprise Application Development
- JB325: JBoss for Advanced Java EE Developers
- Other courses on Hibernate and Seam

Participate in the Linux Community

- Participate in the Fedora Project
- Join a local Linux User Group (LUG)
- Subscribe to topical mailing lists
- Read news or participate in forums at Linux web sites
- Chat with developers and users on IRC

End of Lecture 17

- Questions and Answers
- Summary
  - What to do from here?
    - Further training
    - Community involvement
    - Something else? Explore!